INSTITUTO POLITÉCNICO DE BEJA

Escola Superior de Tecnologia e Gestão

Mestrado em Engenharia de Segurança Informática

Kusko

Sistema baseado em malware para captura e transmissão remota de dados em sistemas Android

Neuza Valente Figueira

Beja

2016

INSTITUTO POLITÉCNICO DE BEJA

Escola Superior de Tecnologia e Gestão

Mestrado em Engenharia de Segurança Informática

Kusko

Sistema baseado em malware para captura e transmissão remota de dados em sistemas Android

Relatório de dissertação de mestrado apresentado
na Escola Superior de Gestão e Tecnologia do Instituto Politécnico de Beja

Elaborado por:
Neuza Valente Figueira

Orientado por:
Professor Doutor Rui Miguel Soares Silva

Beja
2016

Resumo

Malware para Captura de Dados em Sistemas Android

A presente dissertação é relativa à construção de um sistema baseado em *malware* para captura e transmissão remota de dados em sistemas Android, com a finalidade de aplicar uma segurança ofensiva por parte das autoridades ou de outros responsáveis de segurança.

O *malware* irá obter dados pessoais, bem como outras informações relevantes sobre presumíveis criminosos, tais como dados de chamadas telefónicas efetuadas e recebidas, mensagens de texto recebidas e enviadas, fotografias e vídeos e localização GPS, que serão posteriormente enviadas para um servidor remoto.

Será ainda criada uma aplicação de configuração e ocultação do *malware*, que irá permitir a seleção do tipo de dados que este irá obter e também a escolha de um APK onde o *malware* será colocado, de forma a facilitar a sua disseminação.

Aquando da criação do *malware* foi determinado que o nível mínimo da API seria o 16, correspondente à versão **Android 4.1 Jelly Bean**, pelo fato de as versões anteriores já não apresentarem grande visibilidade no mercado.

É ainda de salientar que o *malware* não é detetável pelo utilizador nem afeta o normal funcionamento do sistema.

Palavras-chave: *malware*; Android; captura de dados; transmissão de dados; segurança ofensiva.

Abstract

The present thesis documents the construction of a malware that allows the capture and remote transmission of user data in Android systems, in order to apply an offensive security by the authorities.

The malware will get personal data, and other relevant information about suspected criminals, such as data from incoming and outgoing calls, sent and received text messages, photos and videos and GPS location, which will later be sent to a remote server.

A malware configuration and concealment app will also be created, which will allow the selection of the type of data it will obtain and also the choice of an APK where the malware will be placed, in order to facilitate its dissemination.

It was determined that the minimum API level for the malware to work would be API 16, **Android 4.1 Jelly Bean**, since previous versions no longer exhibit a large share on the market.

It is also important to note that the malware is not detectable by the user nor affect the normal system function.

Keywords: *malware*; Android; data capture; data transmission; offensive security.

Agradecimentos

Em primeiro lugar gostaria de agradecer ao Doutor Rui Silva pela proposta de um desafio que acabou por evoluir e transformar-se nesta dissertação de mestrado.

Um muito obrigado à minha família e amigos, em especial aos meus pais pelo apoio incondicional que sempre me deram.

Ao meu irmão, por ter disponibilizado o seu telemóvel para ser a vítima dos ataques do malware, um obrigado sincero.

A todos os meus mais sinceros obrigados e espero que após esta etapa que agora termina possa de alguma forma retribuir todo o carinho e dedicação com que me têm presenteado. A todos vós dedico este trabalho.

Índice Geral

Índice de Figuras

Índice de Tabelas

Abreviaturas e Siglas

AID – Android ID

API - Application Programming Interface

APK - Android Application Package

CA – Certificate Authority

EXIF – Exchangeable Image File Format

GID – Group ID

IDE – Integrated Development Environment

IMEI – International Mobile Equipment Identity

IPC – Inter-process Communication

PID – Process ID

UID – User ID

URI – Uniform Resource Identifier

1. Introdução

1.1. Contextualização

Cada vez mais estamos dependentes da tecnologia para a execução das nossas tarefas diárias, sejam elas básicas, como a pesquisa de informação, ou mais complexas, como a aquisição de bens e serviços ou o próprio acesso às contas bancárias.

Nos últimos anos, e com o aparecimento e rápida expansão dos *smartphones* no mercado, essa dependência tornou-se ainda mais acentuada, uma vez que estes dispositivos nos permitem ter as comunicações, a Internet, uma câmara, um GPS e muito mais na palma da mão.

No entanto, todo esse potencial que nos é oferecido pela tecnologia vem acompanhado de inúmeros riscos, facto que é comprovado pelo aumento da criminalidade informática nos últimos anos.

A criminalidade informática pode ser definida como qualquer atividade ilegal levada a cabo com recurso a equipamentos informáticos e/ou à internet, pelo que engloba tanto o *download* ilegal de música e filmes como o furto de contas bancárias *online*. O cibercrime inclui, também, ofensas não monetárias, como a difusão de *malware* ou a divulgação de informação confidencial.

O *malware*, isto é, programas informáticos construídos com o intuito de danificar ou efetuar outras ações não desejadas no equipamento informático da vítima, é um dos problemas que tem ganho mais notoriedade, sendo que os exemplos mais comuns de *malware* incluem os vírus, os *worms*, os *trojan horses* e o *spyware*.

No entanto, esse crescimento da criminalidade informática e o aparecimento de novos problemas de segurança têm despoletado quer nas autoridades, quer nos responsáveis de segurança, ou, até mesmo nos utilizadores mais curiosos, a crença de que uma abordagem defensiva, na qual têm de esperar pelos ataques, é totalmente inadequada. Deste modo, verifica-se uma maior preocupação na aplicação de uma segurança de carácter ofensivo.

A segurança ofensiva caracteriza-se por uma abordagem pró-ativa para proteger sistemas, redes e indivíduos, de ataques informáticos.

Assim, e ao contrário da tradicional segurança defensiva que se concentra apenas em identificar e corrigir vulnerabilidades do sistema, a segurança ofensiva reúne esforços na tentativa de descobrir os culpados e de tentar parar ou interromper as operações dos mesmos.

Portanto, um *malware* pode ser utilizado por esses responsáveis de segurança com a finalidade de aplicar uma segurança ofensiva e, é nessa ideia cuja presente dissertação se enquadra.

1.2. Objetivos

Desta forma, pretende-se criar um *malware* para a plataforma Android, com o objetivo de obter dados pessoais, bem como outras informações relevantes sobre presumíveis criminosos, tais como dados de chamadas telefónicas, mensagens de texto, fotografias e vídeos e localização GPS.

Em primeiro lugar, será efetuado um estudo do estado da arte, de forma a identificar e analisar *malware* semelhante ao que se pretende criar ou que partilhe dos mesmos objetivos, e assim, obter novas ideias para o desenvolvimento desta dissertação.

Em seguida, será apresentada uma breve história do sistema operativo Android; as versões do mesmo a considerar e a caraterização dos seus utilizadores.

Posteriormente, será efetuada a apresentação e a explicação detalhada sobre o *malware* criado, os seus objetivos e características.

Por fim, será descrita a fase de testes e as conclusões resultantes dos mesmos, bem como a identificação de possíveis *bugs* ou melhorias.

2. Estudo do Estado da Arte de *Malware* para Android

O estudo do estado da arte é uma das partes mais importantes de qualquer projeto, pois possibilita a descoberta de muita informação sobre o assunto de pesquisa e potencia a melhoria de aplicações já existentes ou o desenvolvimento de novas com base nas já criadas.

Para tal, foi efetuada uma pesquisa exaustiva com o objetivo de determinar a evolução dos diferentes tipos de *malware* para sistemas Android e dos mecanismos de segurança criados para o combater.

2.1. Visão Geral da Arquitetura do Sistema Android

O Android é um sistema operativo para dispositivos móveis baseado em Linux, desenvolvido pela *Open Handset Alliance*, liderada pela Google e outras empresas.

A primeira versão beta do *Android Development Kit* (SDK) foi lançada em 2007, e a primeira versão comercial, o Android 1.0, em Setembro de 2008.

O sistema Android é constituído por componentes que podem ser divididos em cinco seções diferentes: aplicações, Framework, bibliotecas e serviços, Android Runtime e Kernel Linux. A imagem seguinte ilustra a arquitetura do sistema Android: [1]

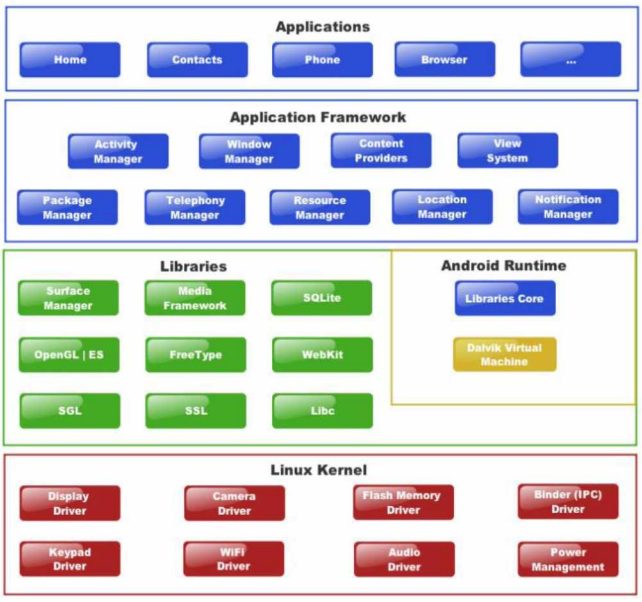

Figura 1 - Arquitetura do Sistema Android

2.1.1. Kernel Linux

O Kernel utilizado no sistema Android é baseado no sistema operativo Linux e é responsável pela camada de abstração entre o *hardware* e as aplicações, bem como pelos serviços principais do sistema, tais como a gestão de processos e memória.

Apesar de algumas funcionalidades do Kernel serem utilizadas diretamente pelo Android, foram necessárias algumas alterações para realizar uma melhor gestão da memória e tempo de processamento, tais como a introdução de um sistema que permite terminar processos de forma criteriosa quando há pouca memória disponível (*Low Memory Killer*) e de um mecanismo de partilha de memória onde vários processos podem comunicar através dessa mesma região de memória (*Ashmem*) [2].

Uma vez que em todos os sistemas operativos que suportam memória virtual os processos são executados em diferentes zonas da memória, ou seja, nenhum processo tem acesso direto à memória de outro processo ou *thread*, é necessária a existência de um mecanismo de comunicação entre processos, que é denominado *Binder*.

2.1.2. Android Runtime

O Android Runtime é composto por uma máquina virtual, denominada *Dalvik VM*, na qual as aplicações são executadas, e por um conjunto de bibliotecas (*core libraries*) que permitem que grande parte das funcionalidades presentes no **Java SE** estejam disponíveis para Android.

A *Dalvik VM* foi criada especificamente para dispositivos móveis, pelo que não executa Java *bytecode* (ficheiros *.class*) diretamente, possuindo uma extensão de ficheiros própria (ficheiros *.dex*). Esses ficheiros são gerados através da transformação das classes Java pelas ferramentas fornecidas pelo Android SDK e encontram-se nos ficheiros JAR ou APK, garantindo um menor consumo de memória.

Para além disso, o Android possui, desde a versão 2.2, uma implementação de *Just-In-Time* que efetua a compilação de *dexcodes* para a versão alvo em tempo de execução, o que torna a execução de processos bastante mais rápida.

2.1.3. Bibliotecas e Serviços

Os serviços do sistema são responsáveis pela implementação da maior parte das funcionalidades básicas do sistema Android, tais como o suporte para o ecrã e *touchscreen*, e conetividade.

As bibliotecas são responsáveis pelas funcionalidades de manipulação de gráficos, áudio, vídeo, bases de dados, entre outras.

De entre as principais bibliotecas destacam-se: **Media Framework**, que disponibiliza diversos *codecs* que permitem a gravação e reprodução de diferentes formatos de média; **OpenGL**, que fornece a interface Java para a renderização de gráficos OpenGL/ES 3D; **SQLite**, utilizada para aceder a dados e efetuar a gestão de bases de dados SQLite; e **WebKit**, mecanismo do *browser* para exibir conteúdo HTML.

2.1.4. Framework

A *Framework* inclui todas as bibliotecas Java que não fazem parte das bibliotecas padrão e fornece os blocos básicos para a construção de aplicações Android, tais como classes para *activities*, serviços e *content providers*, GUI *widgets*, classes para acesso a ficheiros e bases de dados e ainda classes que permitem a interação com o *hardware* do dispositivo.

Alguns dos mais importantes blocos da *Framework* são:

- **Activity Manager**, utilizado para gerir o ciclo de vida das aplicações;

- **Content Providers**, utilizado para gerir a partilha de dados entre duas aplicações;

- **Location Manager**, utilizado para gerir as localizações obtidas através de GPS ou rede móvel;

- **Telephony Manager**, utilizado para gerir todas as chamadas de voz.

2.1.5. Aplicações

As aplicações são os programas com os quais os utilizadores interagem diretamente e, apesar de todas serem construídas com recurso à *Framework*, podem ser distinguidas entre aplicações do sistema e aplicações instaladas pelo utilizador.

As aplicações do sistema estão incluídas na imagem do sistema e não podem ser desinstaladas ou alteradas pelos utilizadores, pelo que são consideradas seguras e possuem mais privilégios que as aplicações instaladas pelos utilizadores.

Estas aplicações podem fazer parte do sistema operativo ou ser apenas aplicações pré-instaladas, como clientes de *email* ou *browsers*.

Os principais componentes das aplicações são [3]:

- **Activities**: uma *activity* é um único ecrã com uma interface gráfica e é a base das aplicações Android.

- **Services**: são componentes executados em segundo plano e que não possuem interface gráfica. São normalmente utlizados para executar operações longas sem bloquear a interface.

- **Content Providers**: fornecem uma forma de acesso aos dados das aplicações que se encontram normalmente em bases de dados ou ficheiros. O acesso aos *content providers* é realizado via IPC e são principalmente utilizados para partilhar dados entre aplicações.

- **Broadcast Receivers**: são componentes que respondem a uma larga escala de eventos, denominados *broadcasts*, que podem ser originados pelo sistema ou por outras aplicações.

2.2. Processo de Boot

Quando o *hardware* é ligado, o código de *boot* da ROM começa a ser executado e carrega o *bootloader* para que este possa continuar o processo.

A imagem seguinte ilustra o processo de *boot* do sistema Android:

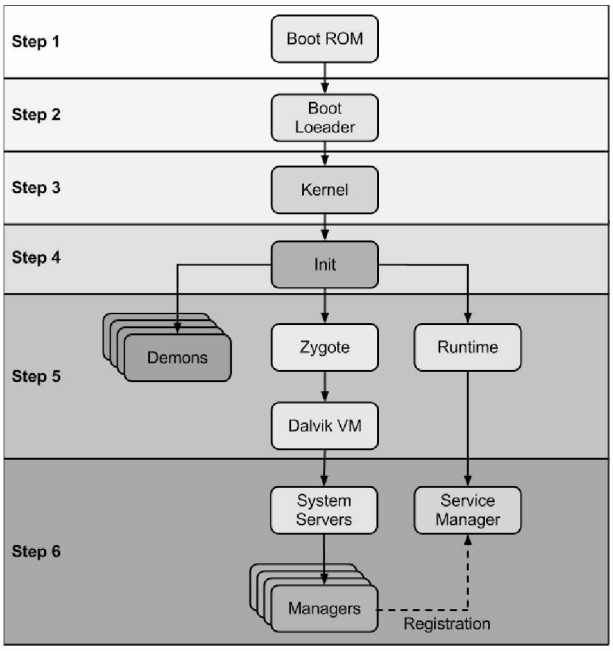

Figura 2 - Ilustração do processo de boot do sistema Android

O *bootloader* é o primeiro componente a ser iniciado e trata-se um pequeno programa que é executado antes do sistema operativo.

A sua execução ocorre em dois passos distintos. No primeiro, o *bootloader* deteta a memória RAM e carrega um programa que irá permitir o segundo passo, onde são configurados a rede, a memória, etc., necessários para a execução do Kernel.

Uma vez terminada a inicialização do *hardware*, o *bootloader* carrega o Kernel da partição do *boot* para a RAM e é este que continua o processo.

O Kernel efetua todas as tarefas necessárias para que o sistema Android seja executado corretamente, tais como a inicialização da memória, *input/output*, proteções de memória, *drivers*, etc. Por fim, monta o sistema de ficheiros *root* e inicia o primeiro processo: o **init** [4].

Este processo é o responsável por inicializar alguns dos serviços mais básicos do Android, tais como o **rild** para telefonia e o **mtpd** para o acesso a VPNs, e o *daemon* **Android Debug Brigde** (adbd). Um desses serviços, denominado **Zygote**, cria a *Dalvik VM* e inicia o primeiro componente Java, o **System Server**. Por fim, outros serviços da *Framework* são inicializados.

2.3. Modelo de Segurança do Android

Tal como o resto do sistema, também o modelo de segurança do Android beneficia das opções de segurança oferecidas pelo Kernel baseado em Linux.

Este sistema operativo utiliza dois modelos de permissões separados, mas que colaboram entre si. Por um lado, o Kernel é responsável por fazer cumprir as permissões com recurso aos utilizadores e aos grupos. Este modelo de permissões é herdado do Linux e é denominado *Android Sandbox*.

O Android Runtime, através da *Dalvik VM* e da *Framework*, aplica o segundo modelo de permissões. Este, que é apresentado ao utilizador no momento de instalação de aplicações, define as permissões das mesmas e limita o seu acesso a recursos do sistema.

2.3.1. *Sandboxing*

O sistema Android atribui automaticamente um UID (*Unique User ID*) exclusivo a cada aplicação no momento de instalação, e executa essa mesma aplicação num processo dedicado que possui o mesmo UID. Para além disso, cada aplicação possui uma diretoria dedicada, na qual apenas essa aplicação possui permissões de leitura e escrita [5].

Assim, as aplicações encontram-se isoladas, ou *sandboxed*, tanto a nível de processos como a nível de ficheiros.

No entanto, e apesar de partilhar o modelo UID/GID do Linux, o sistema Android não possui os tradicionais ficheiros *passwd* e *group* para as credenciais dos utilizadores e grupos.

Em vez disso, o Android define um mapa de nomes com identificadores exclusivos, denominado *Android IDs* (AIDs).

O mapeamento inicial contém entradas reservadas e estáticas para utilizadores privilegiados e/ou críticos do sistema, tal como o utilizador/grupo *system*. Os UIDs para aplicações gerados automaticamente começam no valor 10000 (AID_APP) e os nomes de utilizador correspondentes são escritos na forma app_XXXX, onde XXXX corresponde ao *offset* do AID_APP.

A imagem seguinte mostra um excerto das definições dos AIDs:

```
#define AID_ROOT             0  /* traditional unix root user */

#define AID_SYSTEM        1000  /* system server */

#define AID_RADIO         1001  /* telephony subsystem, RIL */
#define AID_BLUETOOTH     1002  /* bluetooth subsystem */
...
#define AID_SHELL         2000  /* adb and debug shell user */
#define AID_CACHE         2001  /* cache access */
#define AID_DIAG          2002  /* access to diagnostic resources */

/* The 3000 series are intended for use as supplemental group id's only.
 * They indicate special Android capabilities
that the kernel is aware of. */
#define AID_NET_BT_ADMIN  3001  /* bluetooth: create any socket */
#define AID_NET_BT        3002  /* bluetooth: create sco,
                                   rfcomm or l2cap sockets */
#define AID_INET          3003  /* can create AF_INET and
                                   AF_INET6 sockets */
#define AID_NET_RAW       3004  /* can create raw INET sockets */
...
#define AID_APP          10000  /* first app user */

#define AID_ISOLATED_START 99000 /* start of uids for fully
                                    isolated sandboxed processes */
#define AID_ISOLATED_END   99999 /* end of uids for fully
                                    isolated sandboxed processes */
#define AID_USER        100000 /* offset for uid ranges for each user */
```

Figura 3 - Excerto das definições dos AIDs

As aplicações podem ser instaladas com o mesmo UID, chamado *shared user ID*, e neste caso podem partilhar ficheiros e até mesmo ser executadas no mesmo processo. Estes *shared user IDs* são muito utilizados pelas aplicações do sistema, que frequentemente precisam de utilizar os mesmos recursos.

Para além dos AIDs, o Android utiliza grupos suplementares para permitir que os processos acedam a recursos partilhados ou protegidos, ou para conceder direitos adicionais aos processos. Por exemplo, ser membro no grupo **sdcard_rw** permite que o processo consiga executar operações de leitura e escrita na diretoria do cartão de memória.

Quando a aplicação é executada, o seu UID, GID e grupos suplementares são atribuídos ao processo criado. A execução sob um UID e GID únicos permite que o sistema operativo aplique restrições no Kernel e em execução, para controlar as operações entre aplicações.

2.3.2. Sistema de Permissões

Uma vez que as aplicações só conseguem aceder aos seus próprios ficheiros ou a outros recursos acessíveis globalmente, o Android permite a concessão de direitos de acesso adicionais e específicos, para que as aplicações possam funcionar de forma melhorada. Esses direitos de acesso são denominados permissões e controlam o acesso ao *hardware*, Internet, dados ou serviços do sistema operativo [6].

O Android possui um conjunto de permissões pré-definidas e novas permissões relativas a novas funcionalidades são adicionadas em cada nova versão. Atualmente, e no nível de API 24, existem 138 permissões distintas [7].

As aplicações podem solicitar essas permissões através da sua definição no ficheiro **AndroidManifest.xml**. A imagem seguinte mostra o exemplo de uma aplicação que solicita permissões de controlo de Internet e de escrita no armazenamento externo:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Figura 4 - Pedido de permissões por parte de uma aplicação

As permissões são atribuídas a cada aplicação no momento de instalação pelo serviço de gestão de pacotes do sistema. Uma vez concedidas as permissões, as mesmas não podem ser revogadas (exceto se a aplicação for desinstalada) e estão disponíveis para uso da aplicação sem qualquer pedido adicional.

Existem quatro níveis de proteção das permissões, que caracterizam o risco potencial da concessão da permissão: **normal**, ***dangerous***, ***signature*** e ***signatureOrSystem***.

As permissões com nível de proteção normal apresentam pouco risco para o sistema ou para outras aplicações, pelo que são concedidas sem ser necessária a autorização do utilizador. Alguns exemplos de permissões com este nível de proteção são **ACESS_NETWORK_STATE**, que permite que as aplicações obtenham informação sobre redes e **GET_ACCOUNTS**, que permite o acesso à lista de contas existentes no dispositivo.

Por outro lado, as permissões com nível de proteção *dangerous* permitem o acesso aos dados do utilizador ou possibilitam alguma forma de controlo sobre o dispositivo, pelo que o sistema apresenta uma janela de confirmação, contendo informações sobre as permissões solicitadas, no momento da instalação. Alguns exemplos deste tipo de permissão são **READ_SMS**, que permite a leitura das mensagens e **CAMERA**, que permite o acesso à câmara do dispositivo.

Relativamente às permissões com nível de proteção *signature*, estas apenas são concedidas caso a nova aplicação seja assinada com a mesma chave da aplicação que declarou a permissão, para que essa permissão seja exportada. Assim, este é o nível de proteção mais forte, pois implica uma chave criptográfica que apenas o responsável pela aplicação ou plataforma possui.

Estas permissões são geralmente utilizadas por aplicações do sistema que efetuam operações de gestão do dispositivo, e alguns exemplos são **NET_ADMIN**, que permite a configuração de rede, IPSec, etc., e **ACCESS_ALL_EXTERNAL_STORAGE**, que permite o acesso a todo o armazenamento externo existente no dispositivo.

Por fim, as permissões com nível de proteção *signatureOrSystem* são concedidas a aplicações que fazem parte da imagem do sistema ou que estão assinadas com a mesma chave que a aplicação que declarou a permissão. Isto permite que as aplicações pré-instaladas num dispositivo Android partilhem determinadas funções sem a necessidade de partilharem chaves.

O sistema de permissões do Android divide-se em permissões API, permissões do sistema de ficheiros e permissões IPC (*Inter-process communication*).

2.3.2.1. Permissões API

Estas permissões são utilizadas para controlar o acesso a funcionalidades dentro da *Framework*. Um exemplo comum é a permissão **READ_PHONE_STATE**, que garante o direito de leitura do estado do dispositivo. Uma aplicação a que esta permissão seja concedida poderá executar diversos métodos, incluindo métodos presentes na classe **TelephonyManager**, tais como **getDeviceSoftwareVersion** e **getDeviceId**.

Algumas das permissões API correspondem a mecanismos preventivos do Kernel. Por exemplo, a concessão da permissão **INTERNET** traduz-se num pedido de adicionar o UID da aplicação ao grupo **inet**.

2.3.2.2. Permissões do Sistema de Ficheiros

Por defeito, os UIDs e GIDs únicos das aplicações apenas possuem acesso à sua própria diretoria e os ficheiros criados pelas aplicações têm também as permissões de acesso adequadas.

Desta forma, o acesso a outros recursos, como o cartão de memória ou outro armazenamento externo é efetuado através de GIDs. Por exemplo, uma aplicação que solicita a permissão **WRITE_EXTERNAL_STORAGE** irá ter o seu UID adicionado ao grupo que concede direitos de escrita nessa diretoria.

2.3.2.3. Permissões IPC

Estas permissões estão diretamente relacionadas com a comunicação entre aplicações, ainda que se verifique alguma sobreposição com as permissões API.

O Android utiliza uma combinação de um driver do Kernel (o *Binder*) e de algumas bibliotecas para implementar a comunicação entre processos. O *Binder* assegura que o UID da aplicação que a ele recorre não pode ser forjado e vários serviços do sistema dependem desses valores do UID fornecidos pelo *Binder* para controlar o acesso a APIs sensíveis, acessíveis através de IPC. Por exemplo, a permissão **ACCESS_SURFACE_FLINGER**. Esta permissão só é concedida ao utilizador *graphics*, e permite o acesso à interface IPC do serviço de gráficos *Surface Flinger*.

Outro exemplo é o sistema de gestão de Bluetooth, que apenas permite que este seja ativado de forma silenciosa se a aplicação que o solicita possuir o UID do sistema (1000).

Outras permissões que permitem o acesso a todos os métodos de um serviço exposto via IPC podem ser aplicadas pelo sistema através da sua especificação aquando da declaração do serviço, no ficheiro **AndroidManifest.xml**.

Em seguida, o sistema utiliza a base de dados que contém todos os dados de pacotes instalados para determinar quais as permissões requeridas pelo componente que foi chamado (*callee*), obtém um nome do pacote a partir do UID da aplicação que fez a chamada do componente (*caller*) e devolve o conjunto de permissões que foram concedidas a essa aplicação. Caso a permissão requerida pelo *callee* se encontre nesse conjunto de permissões, a execução é efetuada com sucesso, caso contrário, a execução irá falhar e produzir uma *SecurityException*.

2.3.3. Assinatura do Código

Todas a aplicações para sistemas Android, incluindo as aplicações do sistema, devem ser assinadas pelo seu programador.

O Android utiliza a assinatura do APK para garantir que as atualizações de uma aplicação são desenvolvidas pelo mesmo autor e para estabelecer relações de confiança entre as aplicações. Estas medidas de segurança são implementadas através da comparação do certificado de assinatura da aplicação instalada com o certificado de assinatura da atualização ou de uma aplicação a ela relacionada.

O certificado de assinatura é ainda utilizado para controlar as aplicações às quais podem ser concedidas permissões com o nível de proteção *signature*.

A imagem seguinte exemplifica o processo de assinatura de código:

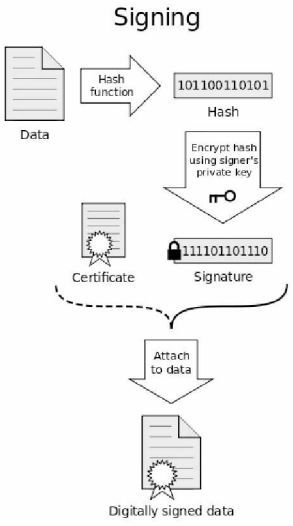

Figura 5 - Assinatura digital das aplicações

O Android utiliza quatro tipos de chaves diferentes: *platform*, *shared*, *media* e *testkey* (ou *releasekey*) [8]. Todas as aplicações que fazem parte do núcleo do sistema (Definições, Telefone, Bluetooth, etc.) são assinadas com *platform keys*. Os pacotes relacionados com a pesquisa ou os contatos são assinados com *shared keys*. A galeria e outras aplicações relacionadas são assinadas com *media keys*. Por fim, todas as outras aplicações são assinadas com *testkeys*.

Uma vez que a assinatura de código em Android é baseada na assinatura do JAR, são utilizados a criptografia de chave pública e certificados X.509. No entanto, o certificado de assinatura não precisa de ser emitido por uma autoridade de certificação, pelo que a grande maioria dos certificados utilizados em Android são *self-signed.*

2.4. Problemas de Segurança

Os mecanismos de segurança do Android possibilitam a existência de um elevado número de vulnerabilidades de segurança, que variam desde o desvio de informação sensível até à execução de código malicioso [9].

2.4.1. Problemas nas Permissões

As permissões formalmente solicitadas por uma aplicação através do seu ficheiro **AndroidManifest.xml** podem não corresponder exatamente às permissões que a aplicação utiliza [10].

Ainda que a documentação descreva a maior parte dos requisitos das permissões para determinados métodos ou classes, esta não é totalmente completa nem precisa.

São vários os exemplos de ações que podem ser efetuadas sem a permissão adequada, ou até mesmo sem nenhuma permissão, ações essas que são bastante valiosas para finalidades maliciosas. Alguns exemplos são: **RECEIVE_BOOT_COMPLETED**, uma permissão que habilita uma aplicação a iniciar com o sistema, ou seja, o *malware* pode iniciar automaticamente e permanecer oculto; **START_ON_INSTALL**: permite que uma aplicação seja iniciada automaticamente após a instalação.

Essas permissões, especialmente quando não são formalmente solicitadas e apresentadas ao utilizador, facilitam a infeção por *malware* e a sua ocultação, uma vez que qualquer aplicação, incluindo aplicações da **PlayStore**, podem servir de camuflagem para *payload* malicioso [11].

Outra das inconsistências encontradas entre a documentação e a implementação foi descoberta na classe **WiFiManager** [12]. Por exemplo, a documentação não menciona as permissões requeridas para o método **startScan**. Isto difere do código fonte do método, que indica uma verificação da permissão **ACCESS_WIFI_STATE**.

Tudo isto resulta em más práticas por partes dos programadores, principalmente através da concessão excessiva de permissões, resultando em graves problemas de segurança, principalmente se essa aplicação for explorada por *malware*.

Aquando da análise de aplicações por permissões excessivas, é importante efetuar uma comparação entre as permissões solicitadas e o propósito da aplicação. Por exemplo, determinadas permissões, como a **CAMERA** e **SEND_SMS** podem ser excessivas para uma aplicação de terceiros, uma vez que essas funcionalidades podem ser obtidas através da delegação das tarefas às aplicações responsáveis pela câmara ou pelas mensagens.

2.4.2. Problemas na Transmissão de Dados Sensíveis

A ideia de segurança na transmissão de informação, através de SSL, TLS, etc., nem sempre é bem utilizada nas aplicações para dispositivos móveis, talvez pela falta de conhecimento de como implementar esses protocolos de forma correta, ou pela noção errada de que as operações efetuadas com recurso à rede da operadora são seguras.

Este problema de segurança pode manifestar-se: através de fraca ou nenhuma encriptação de dados; forte encriptação mas pouco cuidado com erros de validação de certificados de segurança; utilização de texto simples após falhas; e uso inconsistente de mecanismos de segurança dependendo do tipo de rede (rede da operadora ou rede sem fios).

Desta forma, a descoberta de transmissões inseguras é conseguida através de uma simples monitorização do tráfego de rede do dispositivo alvo.

2.4.3. Problemas no Armazenamento de Dados

O Android fornece vários meios de armazenamento de dados, tais como *Shared Preferences* (que permite guardar e obter de dados na forma *key-value*), bases de dados SQLite ou mesmo simples ficheiros.

Desta forma, os erros mais comuns incluem o armazenamento de dados sensíveis sem encriptação e permissões de ficheiros inseguras.

2.5. Estudo da Evolução do Malware

O aumento da popularidade dos *smartphones* Android nos últimos anos, aliado ao fato destes dispositivos armazenarem uma grande quantidade de dados pessoais, resultou no aparecimento de diversos tipos de *malware* para o sistema Android. Para além disso, a natureza *open source* da plataforma Android também é responsável pelo grande número de aplicações maliciosas que surgiram.

Um estudo recente [13] estima que a quantidade total de *malware* para dispositivos móveis tenha crescido a uma taxa de 600% entre Março de 2012 e Março de 2013, sendo que 92% desse *malware* tem como alvo o sistema Android.

Uma das principais vantagens do sistema Android é a sua possibilidade de customização, através da instalação de aplicações disponíveis para *download* em mercados públicos.

Um outro estudo [14] mostrou que o número de aplicações maliciosas na **Google Play Store** cresceu cerca de 400% entre 2011 e 2013, enquanto o número de aplicações maliciosas removidas anualmente pela Google, através de verificações de vírus regulares, desceu dos 60% em 2011 para os 23% em 2013, ainda que o total absoluto de aplicações removidas tenho aumentado das 7000 aplicações maliciosas para as 10000 em 2013.

Para além disso, existem mercados de aplicações não oficiais onde não existe qualquer controlo das aplicações, o que facilita a propagação do *malware*.

Entre 2013 e 2014, voltou a verificar-se um aumento de cerca de 391% no *malware* criado para sistemas Android, o que se traduz em quase um milhão (931.620) de aplicações maliciosas [15].

As próximas seções irão tratar a análise do *malware* e a sua classificação com base no seu comportamento, incluindo instalação, ativação e tipos de *payload*, tendo por base o **Android Malware Genome Project** [16] e o artigo cientifico a ele relacionado [17].

2.5.1. Caraterização do Malware

O **Android Malware Genome Project** é composto por 1260 amostras de *malware*, divididas em 49 famílias diferentes, que foram recolhidas no período de Agosto de 2010 até Outubro de 2011. A tabela relativa à especificação das amostras e locais de recolha das mesmas está disponível no Anexo I.

2.5.1.1. Caraterização segundo o meio de instalação

Este tipo de caraterização baseia-se na forma como o *malware* é obtido e/ou instalado nos dispositivos Android. É importante salientar que estas técnicas não são mutuamente exclusivas uma vez que diferentes variantes da mesma família de *malware* podem utilizar diferentes técnicas para aliciar os utilizadores a obter a aplicação maliciosa.

1. *Repackaging*

Devido ao elevado número de aplicações que são adicionadas à **Play Store** [18] todos os dias, torna-se difícil para as aplicações maliciosas ganharem popularidade suficiente para atingirem um elevado número de *downloads*.

Desta forma, e para evitar essa luta pela popularidade, os autores de *malware* selecionam aplicações legítimas que possuem um elevado número de *downloads*, como o *Facebook* ou o *Netflix*, efetuam engenharia reversa no **APK** para obter o código-fonte, inserem o código malicioso, compilam novamente a aplicação e procedem ao *upload* da nova aplicação maliciosa para os mercados de aplicações.

No total de 1260 amostras de *malware*, 1083 delas (cerca de 86%) são aplicações que passaram por um processo de *repackaging*. É possível verificar que as aplicações selecionadas para *repackaging* são bastante distintas e incluem aplicações pagas, jogos muito populares, utilitários e aplicações relacionadas com pornografia.

Por exemplo, uma das amostras do *malware* **AnserverBot** [19] foi encontrada numa aplicação paga disponível no mercado oficial, **com.camelgames.mxmotor**, e uma das amostras do *malware* **BgServ** [20] numa ferramenta de segurança lançada pela Google para remover o **DroidDream** [21] de dispositivos infetados.

2. *Update*

Apesar de esta técnica também recorrer ao *repackaging* de aplicações populares, é mais difícil de detetar uma vez que, em vez de colocar todo o *payload* na aplicação, inclui apenas um componente de atualização que irá obter o *payload* malicioso aquando da execução da aplicação.

Na amostra de *malware* existem quatro famílias que utilizam esta técnica: **AnserverBot**, **BaseBridge** [22], **DroidKungFuUpdate** [23] e **Plankton** [24].

Por exemplo, quando uma aplicação que contém *malware* da família **BaseBrigde** é executada, é apresentada uma mensagem que informa o utilizador que existe uma nova versão disponível da aplicação. Caso o utilizador aceite, é instalada uma nova versão da aplicação que contém o *payload* malicioso.

3. *Drive-by Download*

Ainda que esta técnica não constitua uma exploração direta de vulnerabilidades dos *browsers* para dispositivos móveis, pretende aliciar o utilizador a descarregar aplicações aparentemente legítimas.

Na amostra de *malware* existem quatro famílias que utilizam esta técnica: **GGTracker** [25], **Jifake** [26], **Spitmo** [27] e **ZitMo** [28]. As amostras de *malware* das famílias **Spitmo** e **ZitMo** foram projetadas para roubar informação bancária sensível.

O **GGTracker** é iniciado a partir dos anúncios publicitários existentes na aplicação, em particular quando o utilizador clica num anúncio específico. Aí, é redirecionado para um *website* que indica estar a analisar a bateria do dispositivo e o redireciona novamente para um mercado de aplicações falso, onde propõe o *download* de uma aplicação de gestão da bateria. Entretanto, o *malware* subscreve um serviço de valor acrescentado sem o conhecimento do utilizador.

4. *Fake Apps*

Este grupo inclui aplicações falsas que se disfarçam de aplicações legitimas e executam ações maliciosas, como o roubo de credenciais de acesso ou o envio de mensagens de texto. No entanto, não são aplicações que passaram por um processo de *repackaging*, são apenas aplicações que replicam a interface original.

Um exemplo deste tipo de *malware* é o **FakeNetflix**, que se faz passar pela aplicação *Netflix* original e obtém os dados de acesso do utilizador.

5. *Hidden Functionality*

Este grupo inclui aplicações que possuem funcionalidades maliciosas propositadamente, mas que também conseguem cumprir a funcionalidade que prometem, não se tratando de um processo de *repackaging* uma vez que estas são novas aplicações criadas para o efeito.

No entanto, executam também operações em segundo plano e sem autorização do utilizador, como o envio de mensagens ou a subscrição de serviços de valor acrescentado.

Um exemplo deste tipo de *malware* é uma das amostras do **RogueSPPush** [29], uma aplicação de astrologia que automaticamente subscreve serviços de valor acrescentado através da ocultação de mensagens de confirmação.

6. Root Exploits

O último grupo inclui aplicações que dependem de privilégios *root* para funcionar corretamente e, caso não solicitem esses privilégios ao utilizador, tiram partido de *exploits* para os obter.

Uma vez obtidos os privilégios *root*, o *malware* pode aceder e modificar definições ou aplicações do sistema, e inclusivamente instalar aplicações em segundo plano, sem o conhecimento do utilizador.

Exemplos deste tipo de malware são o **Asroot** e o **DroidDeluxe** [30].

2.5.1.2. Caraterização segundo o meio de ativação

Este tipo de caraterização baseia-se nos eventos do sistema que o *malware* mais utiliza, uma vez que este pode recorrer a notificações de eventos e chamadas do sistema para executar o seu *payload*.

De todos os eventos do sistema disponíveis, o **BOOT_COMPLETE** é o mais utilizado pelas amostras de *malware*, abarcando 29 famílias (cerca de 83.3% do total de amostras). Este evento ocorre quando o processo de *boot* termina, permitindo assim que o *malware* inicie a execução do seu *payload* com o sistema.

Em segundo lugar, encontramos o evento **SMS_RECEIVED**, que é utilizado por 21 famílias de *malware*. Este fato deve-se à existência de uma grande quantidade de *malware* que visa intercetar ou responder a mensagens de texto.

No entanto, existem alguns tipos de *malware*, como o **AnserverBot** e o **BaseBridge**, que fazem uso de um grande número de eventos. Uma das possíveis explicações prende-se com o fato de que uma maior quantidade de eventos garante a execução de *payloads* maliciosos de forma mais rápida e eficiente.

2.5.1.3. Caraterização segundo o *payload*

Esta caraterização é baseada no tipo de funcionalidade que o *payload* do *malware* possui: elevação de privilégios, controlo remoto, custos financeiros ou recolha de informação.

1. Elevação de Privilégios

O fato de o sistema Android ser constituído por um Kernel baseado em Linux e por uma *Framework* com mais de noventa bibliotecas *open-source* leva ao aparecimento de vulnerabilidades que podem ser exploradas para elevação de privilégios.

No geral, existe um pequeno número de plataformas vulneráveis que são exploradas ativamente. Os quatro principais *exploits* são **asroot**, **Exploid**, **RageAgainstTheCage** (RATC) [31] e **Zimperlich**.

No total de 1260 amostras de *malware*, 463 (cerca de 36.7%) contêm pelo menos um *root exploit*. No entanto, não é incomum que o *malware* utilize vários tipos de *root exploits* para maximizar as suas hipóteses de sucesso, pelo que existem 378 amostras de *malware* que satisfazem essa condição.

A tabela seguinte [17] mostra o número de amostras de *malware* em que foram encontrados os diferentes tipos de *root exploits*:

Vulnerabilidade	*Root Exploit*	N.º de amostras	Exemplo de *Malware*
Kernel	**asroot**	8	**asroot**
init (<=2.2)	**Exploid**	389	**DroidDream, zHash, DroidKungFu**
adbd (<=2.2.1) zygote (<= 2.2.1)	**RATC** **Zimperlich**	440	**DroidDream, BaseBrigde, DroidKungFu, DroidDeluxe**
vold (<=2.3.3)	**GingerBreak**	4	**GingerMaster**

Tabela 1 - Utilização de root exploits pelo malware

2. Controlo Remoto

No total de 1260 amostras, 1172 (93%) possuem a funcionalidade de controlo remoto sobre os dispositivos.

Mas especificamente, 1171 amostras de *malware* utilizam comunicações HTTP para receber comandos dos servidores *Command-and-Control* (C&C), sendo que algumas dessas comunicações são encriptadas.

A maior parte dos servidores C&C estão registados em domínios controlados pelos próprios atacantes. No entanto, foram também identificados casos em que os servidores se encontram alojados em *clouds* públicas.

Exemplos de *malware* que utilizam controlo remoto são **Pjapps** [32], **DroidKungFu3** [33], **Geinimi** [34] e **Plankton**.

3. Custos Financeiros

Uma forma de tornar rentáveis os ataques a dispositivos móveis é através da subscrição de serviços de valor acrescentado controlados pelo atacante, por exemplo através do envio de mensagens. Os principais países afetados por este tipo de ataque são a Rússia, os Estados Unidos e a China.

No total, foram encontradas 55 amostras de *malware* (cerca de 4.4%), pertencentes a 7 famílias diferentes, que enviam mensagens para números de valor acrescentado que se encontram *hardcoded* na aplicação maliciosa.

No entanto, existem algumas amostras que optam por não especificar os números diretamente no código e tiram partido da possibilidade de controlo remoto sobre os dispositivos para inserir os números pretendidos mais tarde. Foram encontradas 13 famílias de *malware* com estas caraterísticas.

Exemplos de *malware* com custos financeiros são **FakePlayer** [35], **GGTracker** e **zSone** [36].

4. Recolha de Informação

O *malware* pode ainda ter como objetivo a recolha de vários tipos de dados dos dispositivos infetados.

No total, foram encontradas 138 amostras de *malware* (13 famílias) que recolhem dados relativos a mensagens de texto, 563 (15 famílias) que recolhem números de telefone e 43 (3 famílias) que recolhem dados de contas de utilizador.

Exemplos de *malware* com o objetivo de recolha de informação são **SndApps**, **FakeNetflix**, **Zitmo** e **Spitmo**.

2.5.1.4. Caraterização segundo as permissões solicitadas

O modelo de permissões do sistema Android é uma boa medida de segurança pois força os programadores a solicitar de forma específica as permissões necessárias para o funcionamento da aplicação.

No entanto, não é possível esperar que os utilizadores conheçam as 138 permissões existentes atualmente nem as implicações de conceder determinadas permissões [37]. Para além disso, aquando da instalação, o utilizador não pode recusar determinadas permissões e aceitar outras, tendo de aceitar todas ou não instalar a aplicação [38].

A tabela seguinte mostra a lista das permissões mais solicitadas pelo *malware*:

Permissão	Amostras	Permissão	Amostras
INTERNET	1232	RECEIVE_SMS	499
READ_PHONES_STATE	1179	VIBRATE	483
ACCESS_NETWORK_STATE	1023	ACCESS_COARSE_LOCATION	480
WRITE_EXTERNAL_STORAGE	847	READ_CONTACTS	457
ACCESS_WIFI_STATE	804	ACCESS_FINE_LOCATION	432
READ_SMS	790	WAKE_LOOK	425
RECEIVE_BOOT_COMPLETED	688	CALL_PHONE	424
WRITE_SMS	658	CHANGE_WIFI_STATE	398
SEND_SMS	553	WRITE_CONTACTS	374

Tabela 2 - Permissões mais solicitadas pelo malware

2.5.2. Ameaças Mais Recentes

Em Novembro de 2015 foi descoberta uma nova ameaça ao sistema Android sob a forma de *adware* e virtualmente impossível de remover. Este *malware* expõe o dispositivo a *root exploits* potencialmente perigosos e oculta-se como uma de várias aplicações, tais como o *Twitter*, *Facebook*, *WhatsApp*, etc. Os investigadores encontraram mais de 20.000 amostras de aplicações fidedignas que sofreram processos de *repackaging* e que foram disponibilizadas em mercados alternativos [39].

A aplicação parece totalmente legítima e executa todas as suas funcionalidades. No entanto, em segundo plano, a aplicação utiliza vários *exploits* para obter acesso *root* sobre o dispositivo. Esses *exploits*, encontrados nas famílias de *malware* **Shedun**, **Shuanet** e **ShiftyBug**, permitem que as aplicações maliciosas sejam instaladas como aplicações do sistema e obtenham altos privilégios, tornando-se assim quase impossíveis de remover, mesmo após o restauro das configurações de fábrica.

A imagem seguinte ilustra o aumento da deteção de amostras do *malware* **Shedun** desde Janeiro de 2016 [40]:

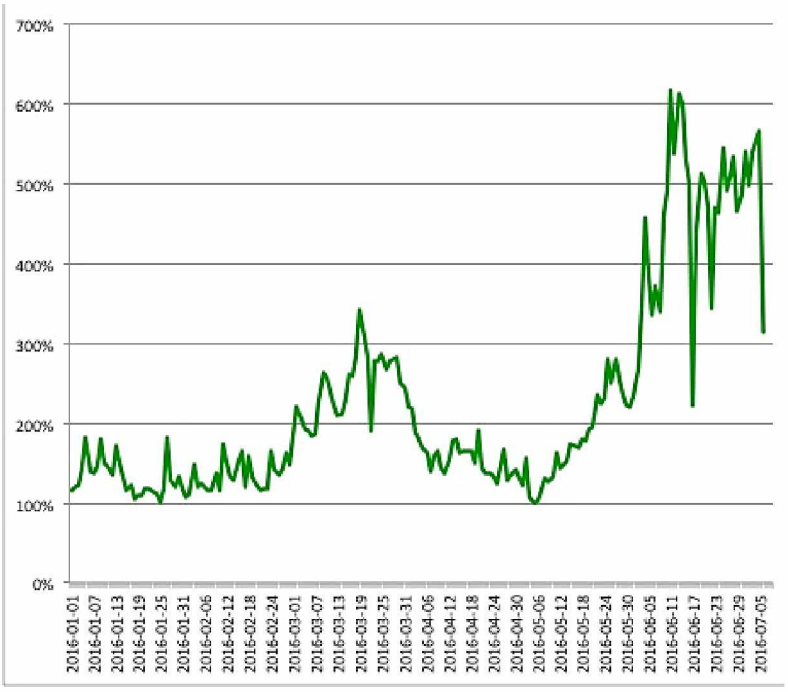

Figura 6 - Aumento nas deteções do malware Shedun desde Janeiro de 2016

Em Fevereiro de 2016 foi descoberta uma nova amostra de *malware*, denominada **HummingBad**, pertencente à família de *malware* **Shedun**.

A imagem seguinte apresenta os vinte países mais afetados pelo **HummingBad**:

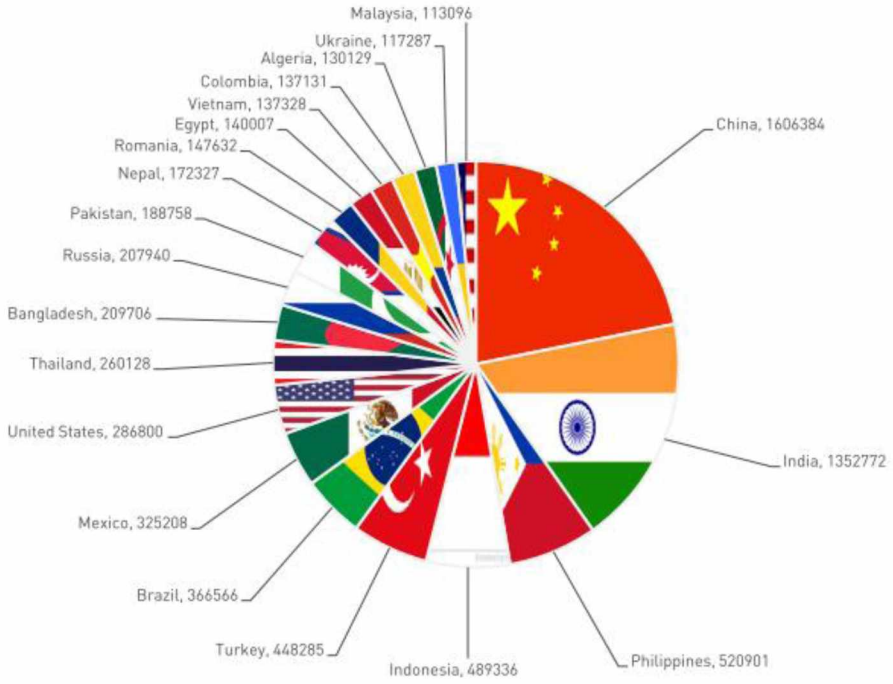

Figura 7 - Países mais afetados pelo HummingBad

Segundo um relatório publicado pela *CheckPoint* [41] relativo ao *malware* **HummingBad**, este gera cerca de $300.000 por mês graças aos 85 milhões de dispositivos Android infetados em todo o mundo. Isto significa que no espaço de um ano, os atacantes conseguem atingir $3.6 milhões em receita, assumindo que o número de dispositivos infetados não continua a aumentar consideravelmente [42]. Para além disso, o *malware* instala ainda aplicações fraudulentas adicionais.

Através dos servidores C&C utilizados pela amostra original do **HummingBad** foi possível determinar a sua origem, uma empresa *ad server* (*web server* que armazena anúncios utilizados em *marketing online*) para dispositivos móveis sediada na China, **Yingmob**.

A tabela seguinte apresenta a quantidade de ataques segundo a versão Android utilizada pelos dispositivos:

Versão Android	Dispositivos Afetados
KitKat (4.4)	50%
JellyBean (4.1.x, 4.2.x, 4.3)	40%
Lollipop (5.0, 5.1)	7%
Ice Cream Sandwich (4.0.3, 4.0.4)	2%
Marshmallow (6.0)	1%

Tabela 3- Quantidade de ataques do HummingBad por versão Android

Ainda que atualmente a motivação dos atacantes seja principalmente financeira, é possível que estes venham a expandir este negócio, por exemplo através da cedência de acesso aos 85 milhões de dispositivos Android que controlam ou através da criação de uma *botnet* para levar a cabo ataques informáticos.

Embora este seja o primeiro grupo a ver exposto ao público o seu elevado nível de organização e autossuficiência financeira, certamente não será o último, uma vez que se prevê um aumento desta perigosa tendência de campanhas de *malware*, com outros grupos a seguirem o exemplo do **Yingmob**.

2.5.3. Previsões

De acordo com um estudo efetuado pela **Webroot** [38], prevê-se o aumento da quantidade de *malware* para dispositivos móveis nos próximos anos, explicado pelo aumento do número desses dispositivos no mercado e pela quantidade de informação que contêm.

No futuro, é esperado um aumento do *malware* de subscrição de serviços de valor acrescentado, uma vez que estes produzem um retorno monetário. Prevê-se também um aumento do *malware* que tem como alvo os serviços bancários, uma vez que também são rentáveis.

Mais recentemente, verifica-se a tendência para o aparecimento de *pop-ups* nos *browsers* de dispositivos móveis que levam o utilizador a pensar que o seu dispositivo está infetado com vírus. Aí, é sugerida a instalação de uma aplicação antivírus que não é mais que uma aplicação maliciosa, que poderá ter custos monetários e/ou aceder a dados sensíveis do utilizador.

Desta forma, prevê-se também o aumento dos *drive-by downloads*, que se irão servir de anúncios nos *websites* ou de vulnerabilidades nos *browsers* para instalar *payloads* maliciosos sem o conhecimento do utilizador.

2.5.4. *Malware* Semelhante

A tabela seguinte apresenta uma lista de *malware* cujas caraterísticas principais são muito semelhantes às do *malware* que se pretende desenvolver, com base em [43]:

Malware	Descrição
Ackposts **Anserver/Answerbot** **BaseBridge** **Booster** **Cosha** **Dougalek** **Kidlogger** **MobileTx** **Sndapps/Snadapps**	Obtém informação pessoal sensível e procede ao seu *upload* para um servidor remoto.
Code4hk/xRAT	Tenta obter a localização da vítima e gravações de voz e envia esses dados para um servidor.
Crusewind	Interceta mensagens de texto e reencaminha-as para um servidor.
DroidJack/SandoRAT	Instala aplicações, tem acesso às mensagens do dispositivo, chamadas de voz, etc.
Flexispy	Monitoriza chamadas de voz, mensagens de texto, atividade *online* e localização.

Tabela 4 - Malware Semelhante

3. Desenvolvimento do Sistema

3.1. Considerações sobre o sistema alvo e utilizadores

Um dos passos fundamentais aquando do desenvolvimento de uma aplicação é a definição do sistema operativo e das versões do mesmo que se pretendem abranger. No entanto, esta não é uma tarefa fácil dada a grande quantidade de sistemas operativos para dispositivos móveis e respetivas versões existentes atualmente. Assim, um dos principais fatores a considerar é o *market share* de cada um dos sistemas operativos, bem como qual ou quais das suas versões são mais utilizadas.

De acordo com dados estatísticos retirados do *site* **NetMarketShare** [44], é possível verificar que no período entre o início de 2016 até ao presente, a distribuição da quota de mercado entre os sistemas operativos para dispositivos móveis (telemóveis e *tablets*) é a seguinte:

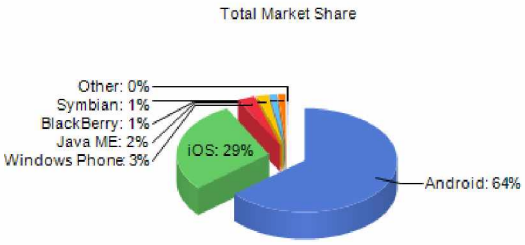

Figura 8 - Market Share dos Sistemas Operativos para Dispositivos Móveis

Uma pesquisa efetuada com base nas versões mais utilizadas do sistema operativo Android no *site* **Statista** [45], no período correspondente ao mês de Maio de 2016, revelou os seguintes resultados:

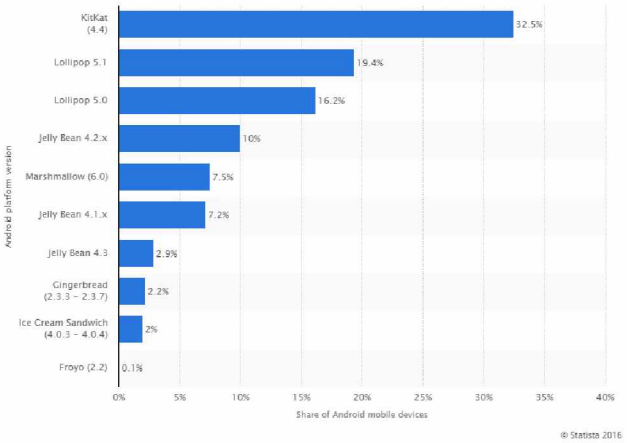

Figura 9 - Market Share das Versões Android

Com base nestas estatísticas, o sistema operativo considerado para o desenvolvimento do *malware* de captura de dados é o sistema Android, versões **4.1 Jelly Bean** até à versão atual, **6.0 Marshmallow**.

Relativamente à caracterização dos utilizadores do sistema, no final de Setembro de 2015, a Google anunciou a existência de 1.4 biliões de *smartphones* com sistema operativo Android ativos [46].

De acordo com estatísticas obtidas num estudo de mercado nos Estados Unidos, efetuado em 2011 [47], foi possível verificar que a grande maioria dos utilizadores do sistema Android se encontram na faixa etária inferior a 35 anos.

A imagem seguinte ilustra a distribuição dos utilizadores por faixas etárias:

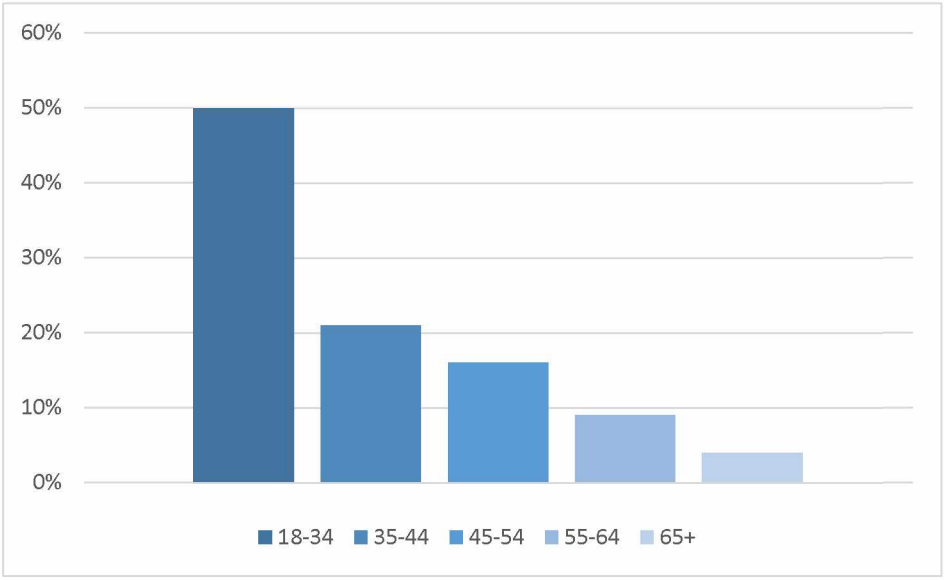

Figura 10 - Distribuição dos utilizadores do sistema Android por grupos etários

Um estudo efetuado pelo *site* **Android Authority** [48] em Dezembro de 2015, com o objetivo de caraterizar o utilizador padrão do sistema Android relativamente ao seu conhecimento do sistema, obteve os resultados apresentados a seguir.

Conhecimento do dispositivo:

Utilizadores que compram produtos de uma determinada empresa de forma ativa conhecem o dispositivo que possuem. Por outro lado, utilizadores que procuram o produto mais barato ou mais recente frequentemente não sabem o nome do produto ou da empresa que o produz.

Conhecimento do *software* e das *features*:

O utilizador regular desconhece a versão de *software* que o seu dispositivo possui, e muitas vezes, desconhece também que o produto pode ser atualizado. Desta forma, e caso não exista qualquer notificação ou caso o processo de atualização não seja automático, a grande maioria dos utilizadores não sabe como atualizar o seu dispositivo.

O utilizador regular não conhece também a grande quantidade de *software* e *hardware* presente no seu dispositivo.

Conhecimento de *Rooting*:

O utilizador regular do sistema Android desconhece como fazer o processo de *root* ao dispositivo, em que consiste ou quais são as suas implicações.

Conhecimento do produto:

O utilizador regular não segue o ciclo de lançamento dos produtos nem está a par de possíveis lançamentos pendentes. De modo geral, o utilizador compra um novo dispositivo quando a sua situação pessoal o dita, por exemplo quando o seu dispositivo atual se estraga.

Um outro estudo [49], realizado em 2012, procurou determinar se o atual sistema de permissões do Android se revela uma medida de segurança eficaz, em especial se os utilizadores estão atentos às permissões apresentadas e se as compreendem.

Foram utilizadas duas amostras, uma com 308 participantes para a realização de um teste *online* e outra com 25 participantes para a realização de um teste de laboratório.

Em ambas as amostras, apenas 17% dos participantes estiveram atentos às permissões apresentadas durante a instalação. Ao mesmo tempo, 42% dos utilizadores do teste de laboratório desconheciam a existência de permissões.

De modo geral, os participantes demonstraram níveis muito baixos de compreensão das permissões, e apenas 3% dos utilizadores do teste *online* responderam corretamente a todas as questões de compreensão.

No entanto, a maioria dos participantes declarou ter recusado a instalação de determinadas aplicações com base em permissões excessivas pelo menos uma vez.

3.2. Arquitetura do sistema

O sistema baseado em *malware* para captura e transmissão remota de dados em sistemas Android pode ser dividido em duas partes: a primeira é relativa à aplicação de configuração e ocultação do *malware* e a segunda refere-se ao próprio *malware* e servidor remoto.

A imagem seguinte ilustra a arquitetura do sistema:

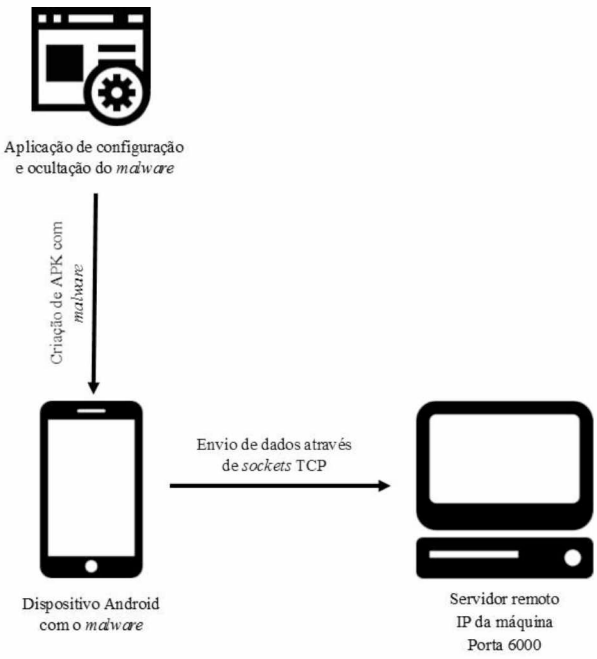

Figura 11 - Arquitetura do sistema

Para o desenvolvimento do *malware* para sistema Android foi utilizada a linguagem **Java** e o ambiente de desenvolvimento utilizado foi o **Eclipse**, com recurso ao **Android Software Development Kit** e ao *plugin* **Android Developer Tools**.

Para o desenvolvimento do servidor, que será executado num computador com sistema operativo Windows, foi utilizada a linguagem de programação **C** e o IDE **Microsoft Visual Studio 2010**.

O envio dos dados entre cliente e servidor será efetuado através de *sockets*. Um *socket* permite que uma aplicação comunique através de uma rede, ou seja, os dados são enviados para o *socket*, este envia os dados através da Internet e esses dados são recebidos por alguém no final da ligação.

Um dos aspetos a ter em consideração para escolher entre um *socket* TCP/IP e um *socket* UDP são as suas características específicas.

O TCP (**Transmition Control Protocol**) trata de todas as conexões entre cliente e servidor e garante que os dados vão chegar ao recetor através do valor **return** do comando **send**.

Por outro lado, o UDP (**User Datagram Protocol**) é *connectionless*, o que significa que a aplicação não tem de passar por uma configuração inicial para se conectar ao servidor ou ao cliente, os pacotes de dados são enviados para um IP e porta sem qualquer informação de que existe alguém no final da ligação para os receber, o que pode levar à perda de pacotes ou à sua receção desordenada.

No que respeita à implementação do *socket*, é necessária a inclusão dos ficheiros **winsock2.h** e **iostream.h** e o recurso à biblioteca **ws2_32.lib**.

O método **CreateSocketServer** é o responsável pela inicialização do *winsock* e pela construção do servidor e suas caraterísticas. O *socket* irá funcionar na porta 6000 e irá receber ligações de qualquer endereço IP, sendo que o endereço IP do servidor será o IP da máquina em que este é executado. Caso se pretenda utilizar o IP público é necessário configurar o encaminhamento de pacotes no *router*.

De forma a garantir uma melhor visualização e análise dos dados obtidos, foi construída uma interface gráfica para o servidor, com recurso ao **C#** e à biblioteca **System.Diagnostics** para iniciar e terminar o processo do servidor quando necessário.

Em primeiro lugar, foi necessário adicionar o executável do servidor aos recursos da interface, para que este esteja incluído na aplicação, evitando assim problemas com o caminho do ficheiro.

Quando a aplicação inicia, é apresentada uma janela que permite ao utilizador selecionar uma pasta onde serão guardados os dados obtidos:

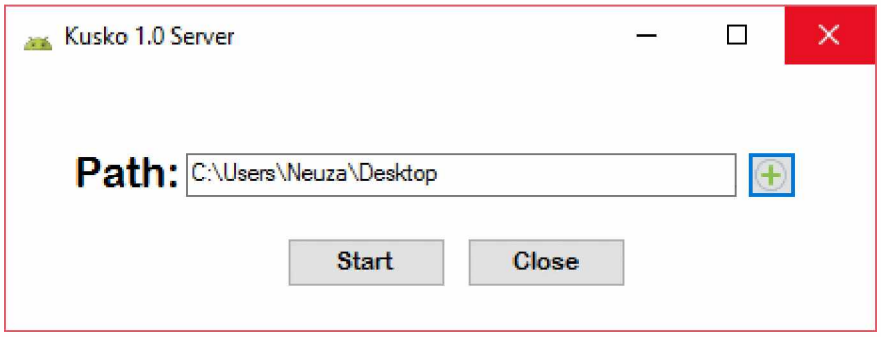

Figura 12 - Interface do Malware: Janela de escolha de pasta

Após a escolha da pasta, é apresentada a janela principal da aplicação:

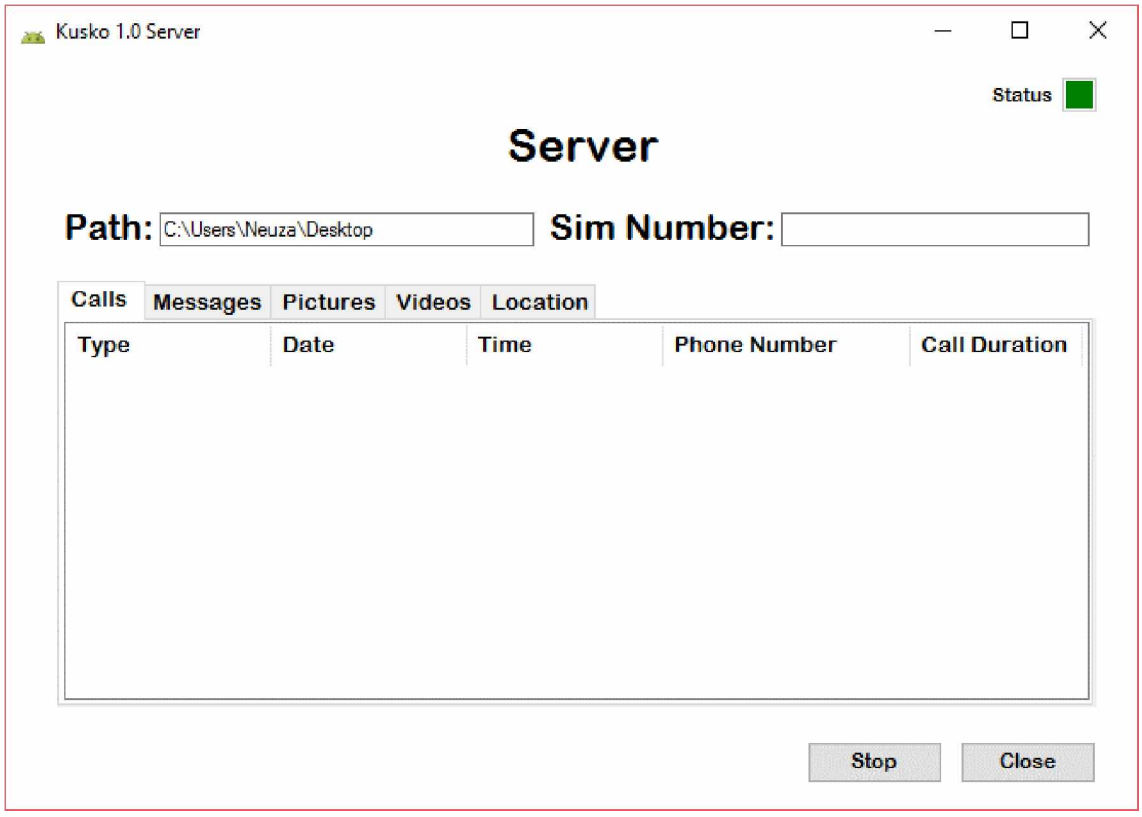

Figura 13 - Interface do Malware: Janela Principal

Esta janela é constituída por vários separadores, de forma a dividir os diferentes tipos de dados obtidos. Cada separador é composto por uma **ListView** com várias colunas, que correspondem às diferentes informações que constituem cada tipo de dados.

A janela apresenta ainda informação relativa ao estado atual do servidor (verde se o servidor se entra ativo, ou vermelho no caso contrário).

3.3. Processamento de dados no *malware*

3.3.1. Chamadas de voz

A obtenção e processamento das chamadas de voz irá dividir-se em duas partes: a primeira refere-se à aquisição dos dados relativos à chamada (tipo de chamada: recebida, efetuada ou perdida, número do contato, data, hora e duração) e a segunda à gravação da mesma para um ficheiro de áudio.

Todas as operações relativas a chamadas de voz são efetuadas através da classe **CallObserver**.

Para a obtenção dos dados foi utilizado um **ContentObserver** para monitorizar alterações ao ficheiro de *log* das chamadas, **CallLog.Calls**.

Um **ContentObserver** [50] é um componente que monitoriza e responde a alterações em determinado conteúdo. Desta forma, cada vez que ocorre uma nova chamada, seja ela uma chamada realizada ou recebida, é criada uma nova entrada no ficheiro de *log* e o **ContentObserver** deteta essa alteração e executa o seu método **onChange**.

Em seguida, é efetuada uma consulta ao ficheiro de *log*, ordenada cronologicamente, e a partir desse resultado são obtidos os dados da última chamada. Para evitar resultados duplicados, é efetuada uma comparação entre o *ID* das chamadas.

Relativamente à gravação das chamadas de voz, esta é efetuada através da utilização de um **MediaRecorder** [51] e de um **BroadcastReceiver** para obter uma notificação quando chamada é iniciada e começar a gravação, e quando a chamada termina para concluir a gravação.

Um **BroadcastReceiver** [52] é um componente responsável por receber e tratar eventos (ou *broadcasts*) provenientes do próprio sistema ou de outras aplicações. Este **BroadcastReceiver** foi criado na classe **LoopService**, que é responsável pela monitorização da atividade do dispositivo em segundo plano, e responde a eventos desencadeados pela ação **ACTION_PHONE_STATE_CHANGED**.

Quando o *broadcast* desta ação é recebido, é efetuada uma verificação do estado atual do dispositivo através da *string* **EXTRA_STATE** da ação acima referida. Se o estado obtido for igual a **RINGING** ou **OFFHOOK**, que corresponde a chamada rece-

bida ou chamada efetuada, respetivamente, é executado o método responsável por iniciar a gravação da chamada.

No entanto, se o estado atual for igual a **IDLE**, que ocorre quando a chamada é terminada, é executado o método responsável por terminar a gravação da chamada.

O resultado da gravação da chamada é um ficheiro **.mp4** cujo nome contém a data e hora da chamada e que será posteriormente enviado para o servidor e eliminado da diretoria onde se encontra.

Permissões:

- *android.permission.READ_PHONE_STATE:* permite o acesso, em modo apenas de leitura, ao estado do telefone, incluindo o número de telefone, informações sobre a rede móvel, o estado das chamadas e a todas as contas registadas no dispositivo.

- *android.permission.PROCESS_OUTGOING_CALLS:* permite que uma aplicação tenha acesso ao número de telefone durante chamadas efetuadas, com a opção de redirecionar a chamada para um outro número ou de cancelar completamente a chamada.

- *android.permission.READ_CALL_LOG:* permite que uma aplicação obtenha acesso ao ficheiro de *log* das chamadas.

- *android.permission.RECORD_AUDIO:* permite que uma aplicação consiga efetuar gravações do áudio do dispositivo em questão.

3.3.2. Mensagens de texto

A obtenção e processamento das mensagens de texto consiste na aquisição dos dados da mesma: tipo de mensagem (enviada ou recebida), número do contato, data, hora e conteúdo.

Todas as operações relativas a mensagens de texto são efetuadas através da classe **SmsObserver**.

Para obter os dados das mensagens de texto foi utilizado um **ContentObserver** para monitorizar alterações no URI "**content://sms/**".

Desta forma, cada vez que se verifica a existência de uma nova mensagem, seja ela enviada ou recebida, é criada uma entrada no conteúdo de "**content://sms/**" e o **ContentObserver** deteta essa alteração e executa o seu método **onChange**.

Em seguida, é efetuada uma consulta ao conteúdo alterado e a partir desse resultado são obtidos os dados da última mensagem de texto. Para evitar resultados duplicados, é efetuada uma comparação entre o *ID* das mensagens.

Permissões:

- *android.permission.READ_SMS:* permite que a aplicação tenha acesso de leitura a todas as mensagens de texto do dispositivo.

- *android.permission.RECEIVE_SMS*: permite que a aplicação tenha acesso às mensagens recebidas.

3.3.3. Fotografias e vídeos

A obtenção e processamento das fotografias e vídeos consiste na aquisição dos dados dos mesmos (tipo de ficheiro: fotografia ou vídeo, data, hora e coordenadas geográficas, estas últimas apenas das fotografias) e do próprio ficheiro.

Todas as operações relativas a fotografias e vídeos são efetuadas através da classe **CameraObserver**.

Para obter os dados das fotografias e vídeos foi utilizado um **FileObserver** para monitorizar alterações na diretoria utilizada pela câmara para guardar os ficheiros.

Um **FileObserver** [53] é um componente que monitoriza uma diretoria e aciona um evento quando ficheiros nessa diretoria são criados, modificados ou eliminados por qualquer processo no dispositivo. Desta forma, cada vez que é capturada uma nova fotografia ou vídeo, esse ficheiro é adicionado à diretoria padrão da câmara e o **FileObserver** deteta essa alteração e executa o seu método **onEvent**.

No entanto, como o **FileObserver** não é recursivo, só seria possível monitorizar uma diretoria específica dentro da pasta DCIM, o que não resultaria caso a aplicação da câmara instalada não fosse a aplicação original. Para resolver esse problema foi implementado um **FileObserver** recursivo com base em [54].

Quando o **FileObserver** deteta o evento **MOVED_TO**, uma vez que a câmara escreve os ficheiros numa localização temporária e posteriormente move-os para a localização final, executa o método **readMediaData**, que recebe como parâmetro o caminho completo do ficheiro que despoletou o evento e é responsável por obter os dados do mesmo.

Relativamente aos dados da localização geográfica onde o ficheiro foi obtido, só é possível aceder aos mesmos caso o GPS se encontre ativo no dispositivo, e estes são obtidos através de dados EXIF [55] presentes na fotografia.

A verificação do estado atual do GPS é efetuada pelo método **gpsEnabled** da classe **DataOperations.**

Permissões:

- *android.permission.READ_EXTERNAL_STORAGE:* permite que a aplicação tenha permissões de leitura no armazenamento externo.

- *android.permission.WRITE_EXTERNAL_STORAGE:* permite que a aplicação tenha permissões de escrita no armazenamento externo.

3.3.4. Localização GPS

A obtenção e processamento da localização do dispositivo consiste na aquisição dos dados de latitude, longitude e data e hora em que as coordenadas foram obtidas.

Todas as operações relativas a localização geográfica são efetuadas através da classe **LocationObserver**, baseada em [56].

Para obter os valores de latitude e longitude foi utilizado um **LocationListener** [57], que monitoriza alterações na localização. O método **getLocation** é o responsável por obter a localização atual do dispositivo, através da rede e/ou através do sinal GPS.

Os dados da localização são obtidos com um intervalo mínimo de um minuto ou com uma distância mínima de dez metros. A alteração da localização do dispositivo despoleta uma nova pesquisa da localização.

Permissões:

- *android.permission.ACCESS_COARSE_LOCATION*: permite o acesso à localização aproximada do dispositivo.

- *android.permission.ACCESS_FINE_LOCATION:* permite o acesso à localização exata do dispositivo.

3.4. Verificação de conetividade sem fios

Uma das funcionalidades do *malware* é o envio dos dados obtidos para um servidor remoto, situação que só se irá verificar caso exista ligação a uma rede sem fios, uma vez que o envio dos dados através da rede móvel está dependente do tráfego disponível.

Para tal, de cada vez que existe uma nova captura de dados, é efetuada uma verificação do estado atual das ligações de rede, através do método **isConnected**, presente na classe **DataOperations.**

Permissões:

- *android.permission.ACCESS_NETWORK_STATE:* permite que a aplicação tenha acesso a informações sobre redes e o estado da ligação atual do dispositivo.

3.5. Envio e receção dos dados

O *socket* cliente para envio dos dados capturados foi criado com recurso à biblioteca **java.net.Socket** e todas as operações relativas ao mesmo encontram-se na classe **ClientThread**.

De cada vez que é necessário enviar dados para o servidor, é criada uma nova *thread* na qual funciona o *socket* cliente, que irá ligar-se ao servidor através do IP do mesmo e da porta 6000.

A classe responsável pelas operações com dados após estes serem capturados é a classe **DataOperations**.

O método responsável pelo envio de dados em formato de texto para o servidor é o método **sendToServer.** Em primeiro lugar, é enviado o tipo de dados a que a restante informação se refere (chamada, mensagem, etc.), para que o servidor a escreva no ficheiro de texto correspondente (cada tipo de dados possui um ficheiro de texto específico). Em seguida, são enviados os restantes dados, sob a forma de uma *string*, com os dados separados pelos caracteres '###'.

O método do servidor responsável por receber as *strings* de dados é denominado **receiveData** e seleciona o ficheiro de texto onde as irá escrever de acordo com o nome do tipo de dados que recebeu do cliente.

Se o tipo de dados corresponder a chamada, fotografia ou vídeo, ou seja, tipos de dados que, para além da informação em formato de texto são ainda constituídos por um ficheiro (de áudio no caso das chamadas, fotografia ou vídeo), é executado o método **sendFileToServer**, responsável pelo envio desses ficheiros para o servidor.

Em primeiro lugar, é enviado o nome do ficheiro em questão, para que o servidor possa criar um novo ficheiro para escrita com o mesmo nome. Em seguida, é enviado o ficheiro, dividido em blocos de 1024 *bytes*, de modo a facilitar a transferência e a evitar a perda de dados no *socket*.

Caso se trate do ficheiro de áudio de uma chamada, este é eliminado da diretoria onde se encontra após ter sido enviado, para não ocupar demasiado espaço no dispositi-

vo e para diminuir as hipóteses de ser descoberto pelo utilizador do dispositivo. O mesmo não se verifica caso o ficheiro for uma fotografia ou vídeo capturado pelo utilizador.

Relativamente ao código do servidor responsável por receber os ficheiros, este é denominado **receiveMedia** e escreve os dados do ficheiro, que recebe do cliente em blocos de 1024 *bytes*, num novo ficheiro com o mesmo nome do ficheiro original, que recebeu previamente.

Permissões:

- *android.permission.INTERNET*: permite que a aplicação crie *sockets* na rede.

3.6. Escrita de dados em ficheiro

Uma das funcionalidades do *malware* para garantir que não existe perda de informação caso o dispositivo não esteja ligado a uma rede sem fios no momento da captura é a escrita dos dados obtidos num ficheiro de texto. Esse ficheiro será guardado numa diretoria oculta do armazenamento externo até que se verifiquem todas as condições para o envio dos dados para o servidor.

O método responsável por esta funcionalidade é o método **writeToFile** da classe **DataOperations**, que escreve os dados no ficheiro de texto sob a forma de uma *string*, com os dados separados pelos caracteres '###'.

Quando a ligação a uma rede sem fios é estabelecida, verificação efetuada pelo método **isConnected** referido anteriormente, e caso exista um ficheiro de texto com dados que não puderam ser enviados, é executado o método **readFromFile**, responsável por ler os dados do ficheiro de texto linha a linha e proceder à chamada do método responsável pelo envio dos mesmos para o servidor.

Permissões:

- *android.permission.WRITE_EXTERNAL_STORAGE:* permite que a aplicação tenha permissões de escrita no armazenamento externo.

3.7. Ativação do *malware* com o sistema

De forma a assegurar a monitorização constante do dispositivo, mesmo após este ser reiniciado, foi necessário garantir que o *malware* inicia com o sistema.

Para tal, foi criado um **BroadcastReceiver**, denominado **AutoStart**, que responde a eventos desencadeados pela ação **BOOT_COMPLETED**, que ocorre quando o processo de *boot* do dispositivo termina, e que procede posteriormente à inicialização do serviço de monitorização principal do *malware*, **LoopService**.

Permissões:

android.permission.RECEIVE_BOOT_COMPLETED: permite que uma aplicação receba *broadcasts* da ação **ACTION_BOOT_COMPLETED**, que ocorre quando o sistema termina a sua inicialização.

3.8. Aplicação de configuração e ocultação do malware

A configuração do *malware* consiste na seleção dos dados que se pretendem obter: todos os dados possíveis, apenas dados de chamadas, de mensagens, fotos e vídeos ou localização.

Outro dos aspetos da configuração é a escolha do endereço IP e porta do servidor, caso o utilizador não pretenda utilizar os valores padrão.

Por fim, o programa de configuração fornece um APK, ou permite que o utilizador selecione um à sua escolha, que será utilizado para ocultar o *malware*.

De acordo com as opções selecionadas pelo utilizador, o programa procede à alteração do código fonte do *malware* para eliminar as funcionalidades que não serão utilizadas. Uma vez que a classe **LoopService** é a responsável por todos os serviços de monitorização específicos, é suficiente eliminar desta classe as linhas de código que contêm a inicialização dos serviços que o utilizador não selecionou.

Da mesma forma, caso o utilizador selecione um endereço IP e uma porta diferentes dos valores padrão, o programa irá alterar o código fonte do *malware*, colocando os valores pretendidos nas linhas correspondentes da classe **ClientThread**.

De forma a facilitar a seleção de todas estas opções, foi criado um pequeno programa em Java, com a interface apresentada a seguir:

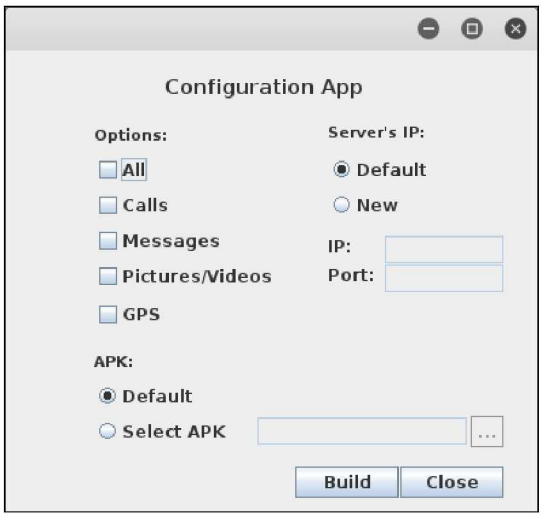

Figura 14 - Interface da aplicação de configuração do malware

Para garantir que existe sempre uma cópia do código original do *malware*, cada execução da aplicação de configuração efetua uma cópia do mesmo e trabalha apenas nesse código.

Relativamente à ocultação do *malware*, a mesma foi realizada com recurso ao sistema operativo **Kali Linux** e à linguagem de programação **Python**, resultando num *script* que será executado a partir do programa de configuração. Este *script* foi realizado com base em [58] e é denominado **ApkBackdoor**.

Após a escolha do APK por parte do utilizador, é utilizada a ferramenta **Apktool** [59] para o descompilar e converter em **Smali Code** [60].

No entanto, e apesar de ser possível alterar o código descompilado de forma a que este execute o *malware*, este código será sempre um pouco diferente entre APKs, pelo que se torna difícil efetuar essa alteração de forma automática. Assim, em vez de utilizar o código descompilado para ocultar o *malware*, será utilizado o código do *malware* para ocultar a aplicação descompilada.

Para o utilizador, a aplicação final irá ter o aspeto e funcionalidade originais, bem como as funcionalidades ocultas do *malware*. Desta forma, torna-se possível utilizar a presente técnica de ocultação em qualquer APK, independentemente do seu código.

De forma a executar o código da aplicação automaticamente quando o *malware* inicia, mas garantindo que esta funciona normalmente, é necessário adicionar o código malicioso ao método que é chamado quando a aplicação inicia, tipicamente o método **onCreate** da atividade principal.

Para tal, é necessário pesquisar o ficheiro **AndroidManifest.xml** da aplicação, que se tornou acessível após esta ter sido descompilada, e procurar as linhas seguintes:

<category android:name="android.intent.category.LAUNCHER" />

<action android:name="android.intent.action.MAIN" />.

De modo a efetuar esta pesquisa de forma automática, é necessário analisar o ficheiro com recurso a uma **ElementTree** [61], que proporciona uma representação em árvore de todo o documento XML.

Quando as linhas pretendidas são encontradas, é tomada a primeira medida para o processo de ocultação do *malware*: a alteração da atividade principal da aplicação, que será substituída pela atividade principal do *malware*. Para tal, é eliminado do **Manifest** o **intent-tag** que indica a atividade principal, que será posteriormente substituído pelo **intent-tag** correto.

Uma vez encontrada a atividade principal da aplicação descompilada é necessário iniciá-la automaticamente, através da adição das seguintes linhas de código ao método **onCreate** da atividade principal do *malware*:

Intent intent = new Intent(getApplicationContext(), <appmain>);

startActivity(intent);

Esta alteração do código do *malware* é efetuada com recurso a **Regular Expressions** [62].

De forma a garantir que o código do *malware* seja compilado sem erros, é necessário que a classe que irá ser iniciada através do **Intent** anterior exista.

Para isso, será criada uma classe com o nome correto, mas que contenha apenas o código essencial à compilação, uma vez que será posteriormente substituída pela classe correta.

O passo seguinte é a compilação do código do *malware*, através do Android SDK e da ferramenta **Ant** [63].

Uma vez compilado o código do *malware*, é necessário voltar a descompilá-lo para o converter em **Smali**, recorrendo novamente ao **Apktool**. Em seguida, o código **Smali** do *malware* é copiado para a pasta que contém o código **Smali** do APK escolhido pelo utilizador.

Para terminar o processo de ocultação do *malware* é necessário adicionar as permissões necessárias ao seu funcionamento ao **AndroidManifest** da aplicação principal, novamente através de **Regular Expressions**.

Antes de compilar o código final é necessário lidar com um pequeno *bug* do **Apktool**: por vezes ocorre um erro de sintaxe aquando da descompilação do ficheiro **Styles.xml** (*@android* torna-se *@*android*).

Visto que se trata de código **Smali** e não **Java**, é necessário utilizar o **Apktool** para compilar a aplicação final.

Após a compilação, é necessário proceder à assinatura do novo APK, uma vez que o sistema Android assim o requer. Para tal, é utilizada a ferramenta **JarSigner** [64].

Para assinar um APK com uma *release key*, é necessário criar previamente uma *keystore*, repositório onde as chaves privadas e certificados são armazenados, na diretoria **/root/.android**.

Em seguida, o APK é assinado com recurso à *keystore* criada previamente. Durante a execução, será necessário inserir a *Passphrase* de *keystore*, definida aquando da sua criação.

Por fim, é efetuado o **Zipalign** do APK [65], através da ferramenta com o mesmo nome, de modo a otimizar a aplicação e a diminuir o consumo de RAM.

Este *script* é executado através do programa de configuração, e recebe como argumentos a localização do APK escolhido pelo utilizador e *password* **root** do sistema operativo, uma vez que algumas das ações efetuadas pelo *script* requerem esse tipo de permissões.

4. Avaliação do sistema

4.1. Configuração e ocultação do malware

Para a preparação dos testes da ocultação do *malware* foi utilizada uma máquina virtual com o sistema operativo **Kali Linux** de 64 bits, com 2GB de memória RAM.

O dispositivo utilizado para o teste das aplicações após configuração e ocultação do malware foi um **Meo Smart A80**, com processador Quad-Core a 1.3GHz, 1GB de RAM e sistema operativo **Android 5.0 Lollipop**.

Os APKs onde o *malware* será ocultado foram escolhidos de forma aleatória e obtidos da **Play Store** através do **APK Downloader** [66].

4.1.1. Resultados

4.1.1.1. APK padrão: Duck Duck Go Search & Stories

O APK padrão da aplicação de configuração é o **Duck Duck Go Search & Stories** [68], um motor de busca que não efetua qualquer tipo de monitorização.

O APK original apresenta as seguintes características:

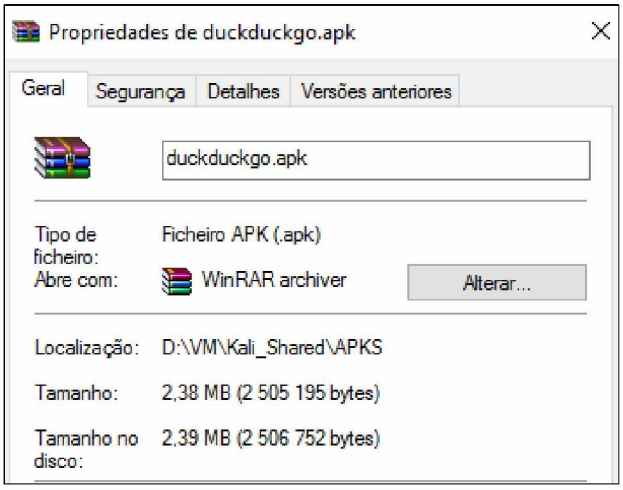

Figura 15 - Características do APK Duck Duck Go Original

Através da aplicação de configuração é executado o *script* que irá efetuar todos os passos necessários para a ocultação do *malware* já referidos anteriormente.

O APK resultante apresenta as seguintes características, verificando-se apenas um aumento insignificante do seu tamanho:

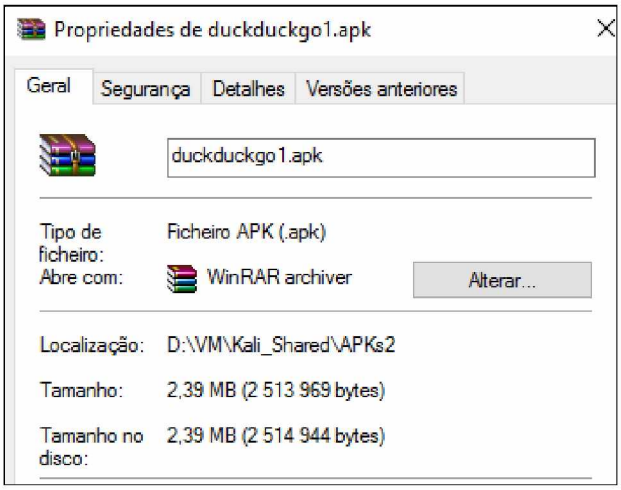

Figura 16 - Características do APK Duck Duck Go após Ocultação do Malware

A verificação de vírus no APK malicioso efetuada através do *site* **VirusTotal** [69] obteve os seguintes resultados:

Figura 17 - Verificação de Vírus no APK Duck Duck Go malicioso

Como se pode verificar, apenas 1 dos 57 antivírus (**Bkav** [69]) conseguiu identificar o *malware* presente no APK, o que denota que a alteração efetuada no APK foi mínima e demonstra que este *software* se encontra obsoleto no que respeita à deteção de *malware* para dispositivos móveis.

As imagens seguintes mostram a instalação do APK malicioso e o seu normal funcionamento, com todas as funcionalidades que o **Duck Duck Go** original oferece:

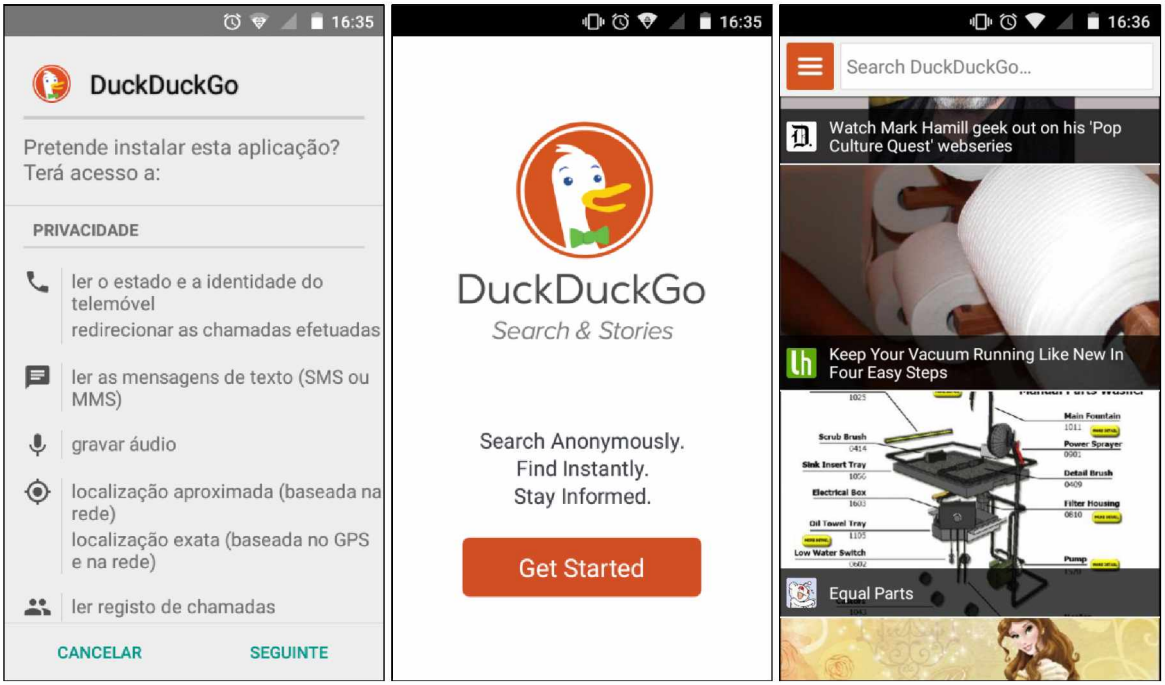

Figura 18 - Screenshots da Instalação e Funcionamento Normal do APK malicioso Duck Duck Go

No entanto, ao efetuar a verificação das aplicações em execução no dispositivo, é possível verificar que a aplicação **Duck Duck Go** iniciou o serviço principal de monitorização do *malware*, **LoopService**, como é possível verificar na imagem seguinte:

Figura 19 - Serviço em execução no APK Duck Duck Go malicioso

4.1.1.2. Instagram

Um dos APK utilizados para ocultação do *malware* foi o **Instagram** [70]. A imagem seguinte apresenta as características do APK original e do APK malicioso, verificando-se novamente um ligeiro aumento no tamanho:

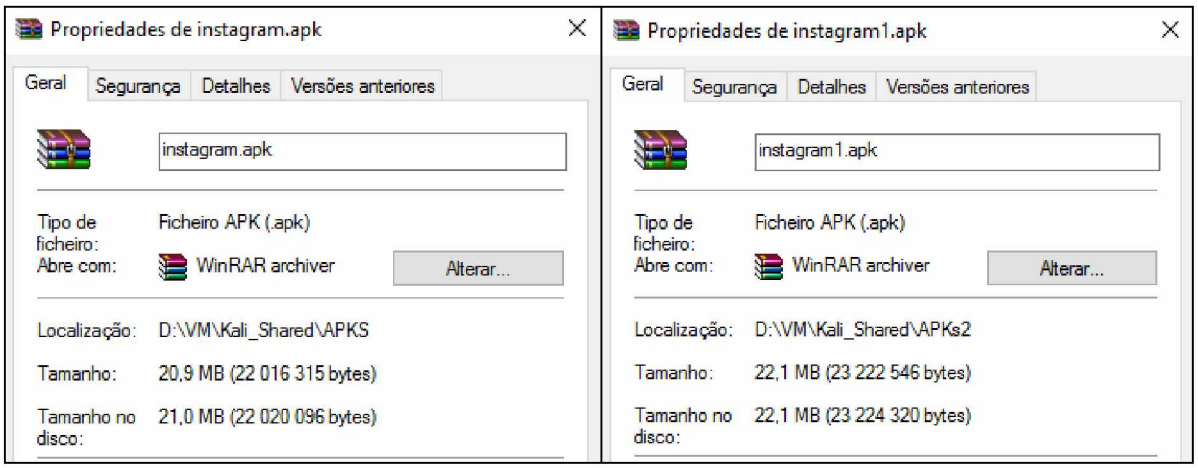

Figura 20 - Comparação entre as Características do APK Instagram original e APK Instagram Malicioso

Foi novamente efetuada uma verificação de vírus no APK malicioso através do *site* **VirusTotal**, que obteve os seguintes resultados:

Figura 21 - Verificação de Vírus no APK malicioso Instagram

Como é possível verificar, a alteração ao APK do **Instagram** foi detetada por dois antivírus distintos (**Bkav** e **AVG**).

As imagens seguintes mostram a instalação do APK malicioso e o seu normal funcionamento, com todas as funcionalidades que o **Instagram** original oferece.

Na última das imagens é possível verificar que a aplicação iniciou o serviço principal de monitorização do *malware*, **LoopService**:

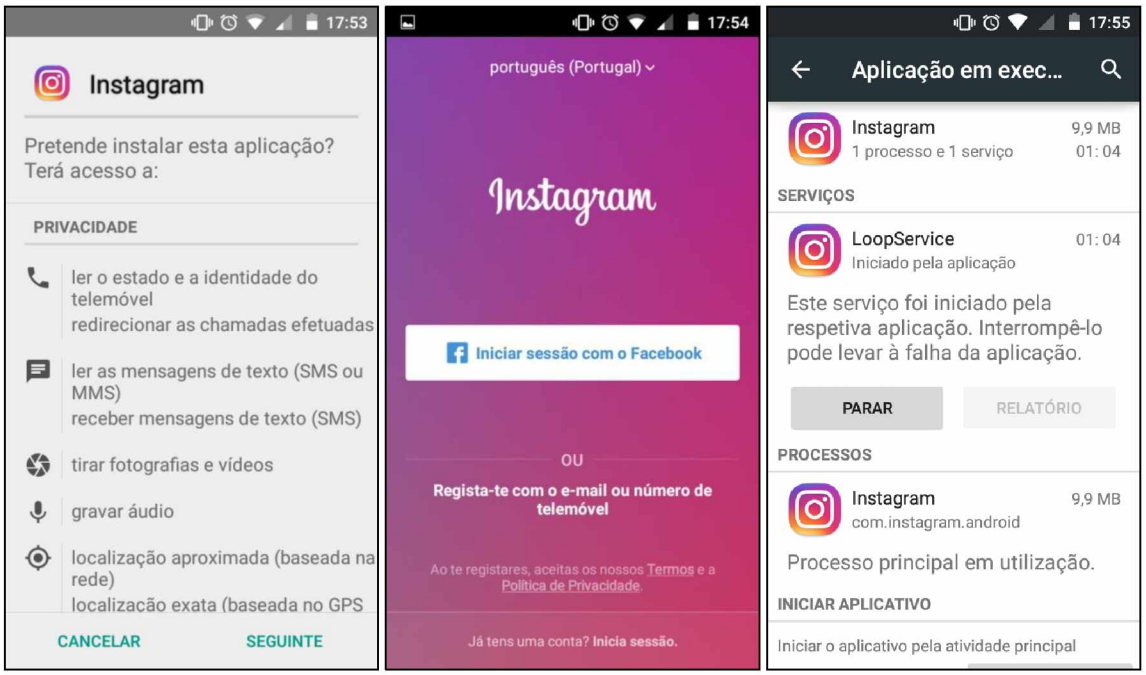

Figura 22 - Screenshots da Instalação e Funcionamento Normal do APK Instagram malicioso

4.1.1.3. Opera Mini

Outro dos APK utilizados foi o *browser* **Opera Mini** [71]. A imagem seguinte apresenta as características do APK original e do APK malicioso, que mostra mais uma vez um pequeno aumento no tamanho:

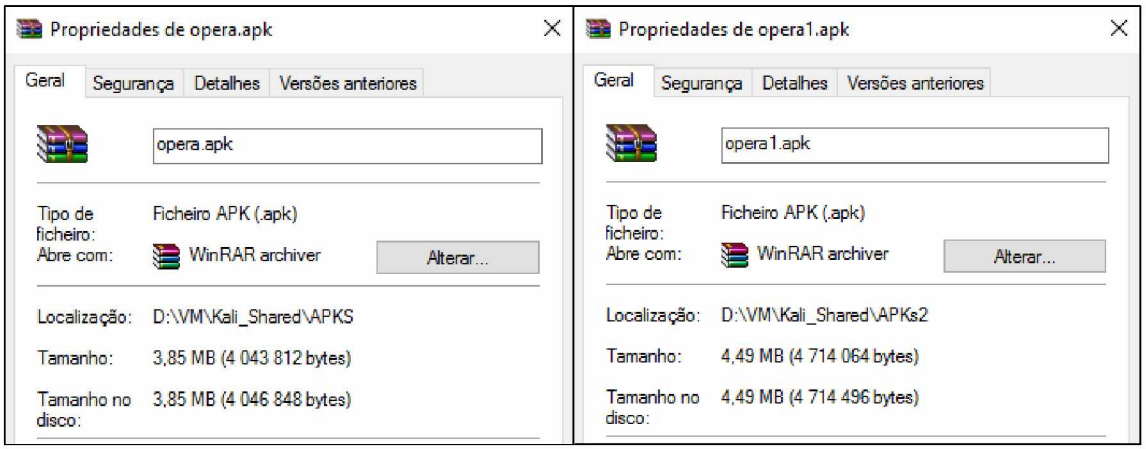

Figura 23 - Comparação entre as Características do APK Opera Original e APK Opera Malicioso

Uma verificação de vírus ao APK malicioso através do *site* **VirusTotal** obteve os seguintes resultados:

Figura 24 - Verificação de Vírus no APK Opera malicioso

As imagens seguintes mostram a instalação do APK malicioso e o seu normal funcionamento, com todas as funcionalidades que o **Opera Mini** original oferece. Na última das imagens é possível verificar que a aplicação iniciou o serviço principal de monitorização do *malware*, **LoopService**:

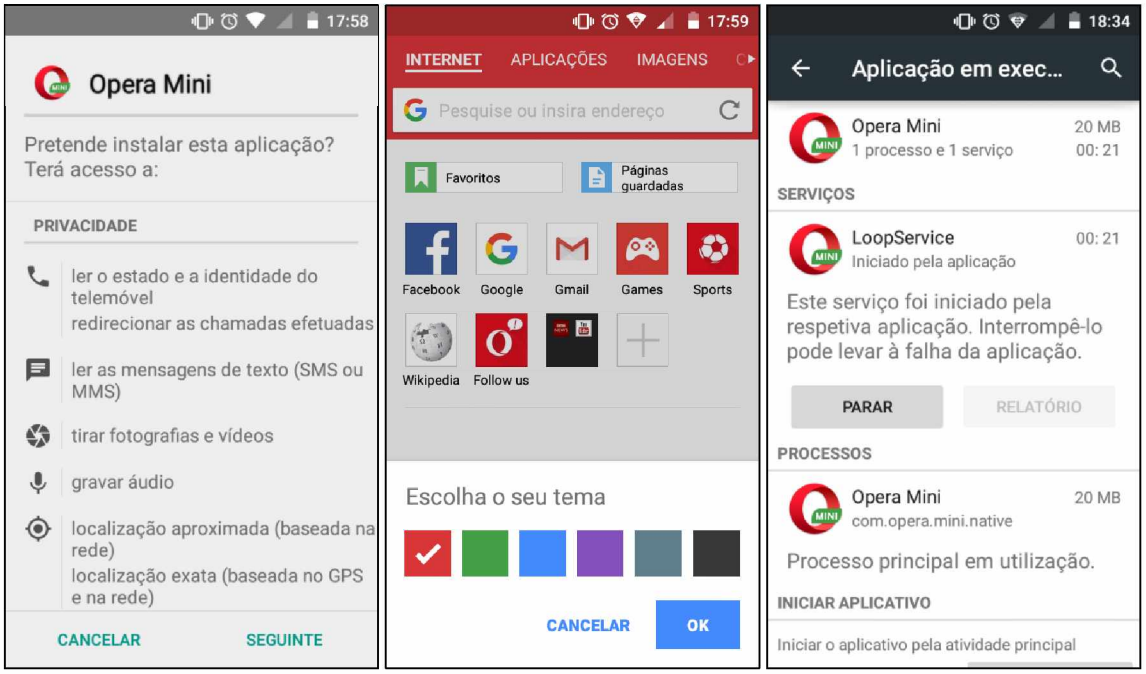

Figura 25 - Screenshots da Instalação e Funcionamento Normal do APK malicioso Opera Mini

4.1.1.4. Candy Crush Saga

Foi também utilizado o APK do jogo **Candy Crush Saga** [72] para ocultar o *malware*. A imagem seguinte apresenta as características do APK original e do APK malicioso:

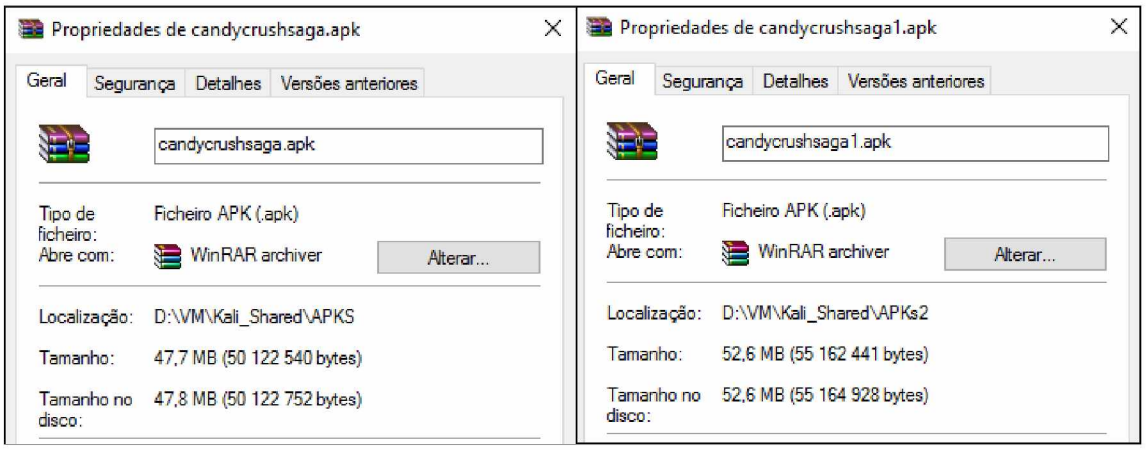

Figura 26 - Comparação entre as Características do APK Candy Crush original e APK Candy Crush Malicioso

O aumento de tamanho mais notório neste APK deve-se à grande quantidade de imagens e outros recursos que este contém, que sofreram um nível de compressão diferente do original.

A verificação de vírus através do *site* **VirusTotal** obteve novamente apenas uma deteção, como é possível verificar na imagem seguinte:

Figura 27 - Verificação de Vírus no APK Candy Crush Saga malicioso

As imagens seguintes mostram a instalação do APK malicioso e o seu normal funcionamento, com todas as funcionalidades que o **Candy Crush Saga** original oferece. Na última das imagens é possível verificar que a aplicação iniciou o serviço principal de monitorização do *malware*, **LoopService**:

Figura 28 - Screenshots da Instalação e Funcionamento Normal do APK Candy Crush Saga malicioso

4.1.1.5. Arrow Launcher

Por fim, foi utilizado o APK do **Arrow Launcher** [73] para ocultar o *malware*. A imagem seguinte apresenta as características do APK original e do APK malicioso:

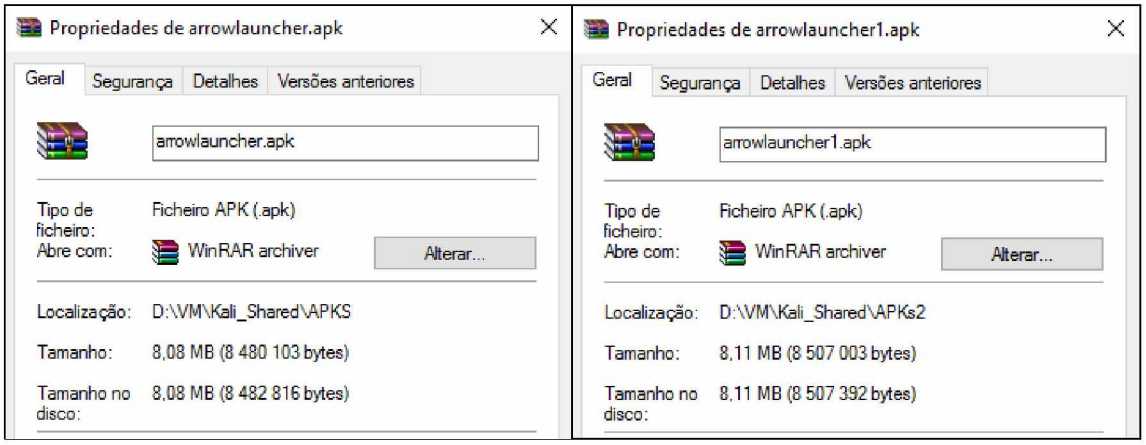

Figura 29 - Comparação entre Características do APK Arrow Launcher original e APK Arrow Launcher Malicioso

Foi mais uma vez efetuada uma verificação de vírus no APK malicioso através do *site* **VirusTotal**, que obteve os seguintes resultados:

Figura 30 - Verificação de Vírus no APK Arrow Launcher malicioso

As imagens seguintes mostram a instalação do APK malicioso e o seu normal funcionamento, com todas as funcionalidades que o **Arrow Launcher** original oferece. Na última das imagens é possível verificar que a aplicação iniciou o serviço principal de monitorização do *malware*, **LoopService**:

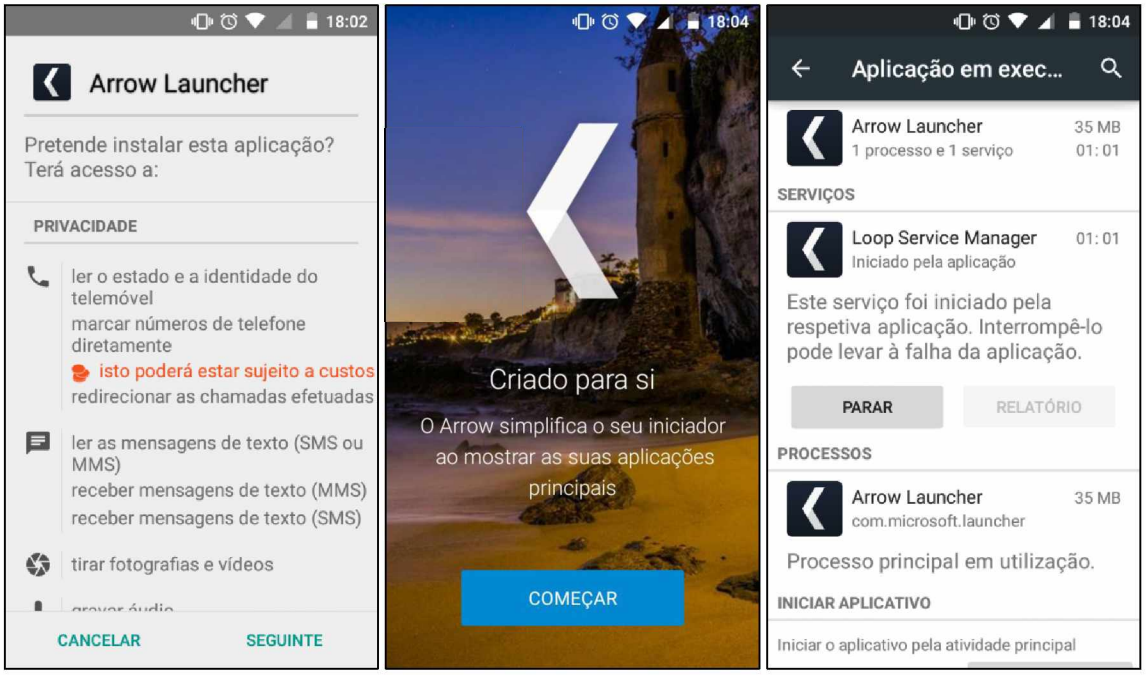

Figura 31 - Screenshots da Instalação e Funcionamento Normal do APK malicioso Arrow Launcher

4.2. Malware e Servidor

Para a preparação dos testes do *malware* foi utilizado um *smartphone* **Meo Smart A80**, com processador Quad-Core a 1.3GHz, 1GB de RAM e sistema operativo **Android 5.0 Lollipop**.

Para os testes do servidor foi utilizado um computador com sistema operativo **Windows 10** de 64 bits, com 8GB de RAM.

Uma vez que ambos os dispositivos se encontram ligados à mesma rede, o IP utilizado para conexão ao servidor é o IP privado da máquina onde o servidor é executado.

4.2.1. Resultados

4.2.1.1. Chamadas de Voz

O primeiro teste efetuado pretende demonstrar o funcionamento da captura e envio dos dados relativos a chamadas de voz.

Para tal, foram efetuadas três chamadas diferentes: uma chamada recebida, uma chamada efetuada e uma chamada perdida. A imagem seguinte mostra o resultado dessas chamadas no *smartphone*:

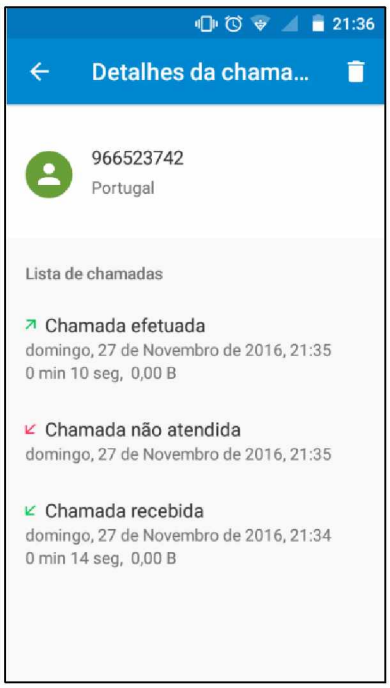

Figura 32 - Resultado das chamadas de voz no smartphone

Relativamente ao servidor, todas as chamadas de voz processadas pelo *smartphone* foram enviadas para este, como é possível verificar na imagem seguinte:

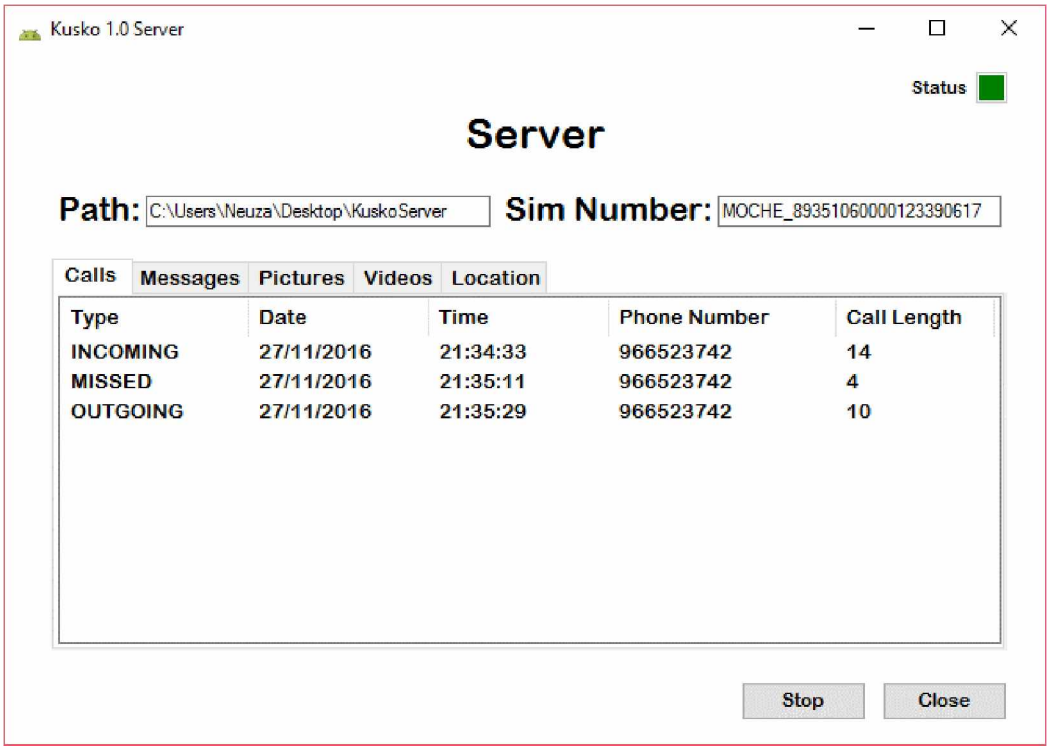

Figura 33 - Resultado das chamadas de voz no servidor

Por sua vez, todos os ficheiros de áudio relativos às chamadas encontram-se na pasta correspondente, como mostra a imagem seguinte:

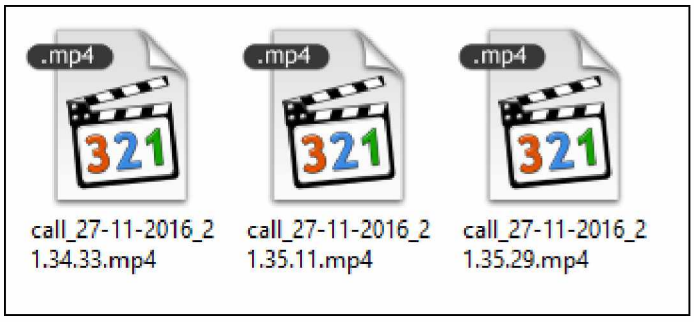

Figura 34 - Ficheiros de áudio das chamadas

Para a sua reprodução através da interface gráfica do servidor, é necessário o duplo clique numa das chamadas apresentadas na lista.

Quando tal acontece, é apresentada a janela seguinte, que possui um objeto **Windows Media Player** para a reprodução dos ficheiros:

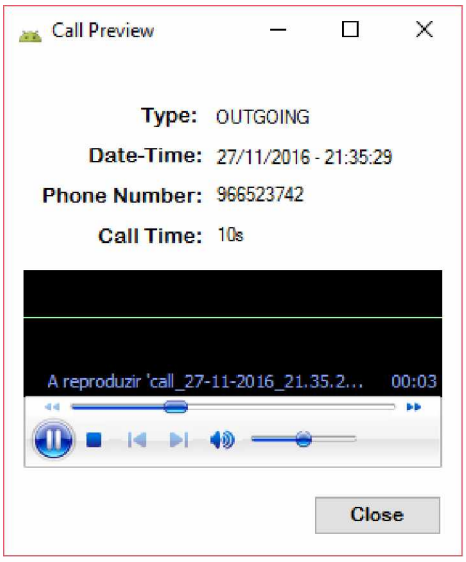

Figura 35 - Reprodução dos ficheiros de áudio das chamadas

4.2.1.2. Mensagens de Texto

Relativamente ao segundo teste, este prende-se com a demonstração do funcionamento da captura e envio de dados relativos a mensagens de texto, tendo sido criadas duas mensagens diferentes: uma mensagem enviada e uma mensagem recebida.

A imagem seguinte mostra o resultado dessas mensagens no *smartphone*:

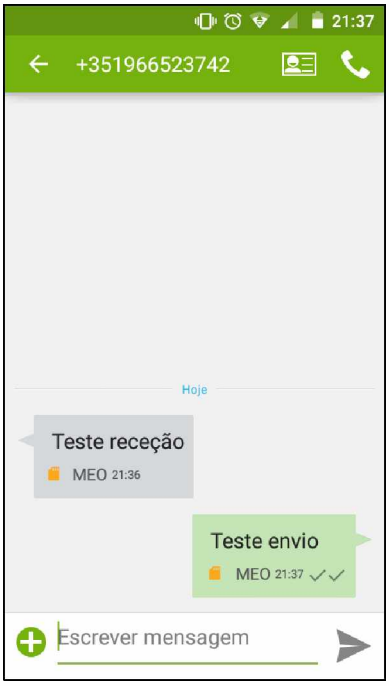

Figura 36 - Resultado das mensagens de texto no smartphone

Relativamente ao servidor, todas as mensagens de texto processadas pelo *smartphone* foram enviadas para este, como é possível verificar na imagem seguinte:

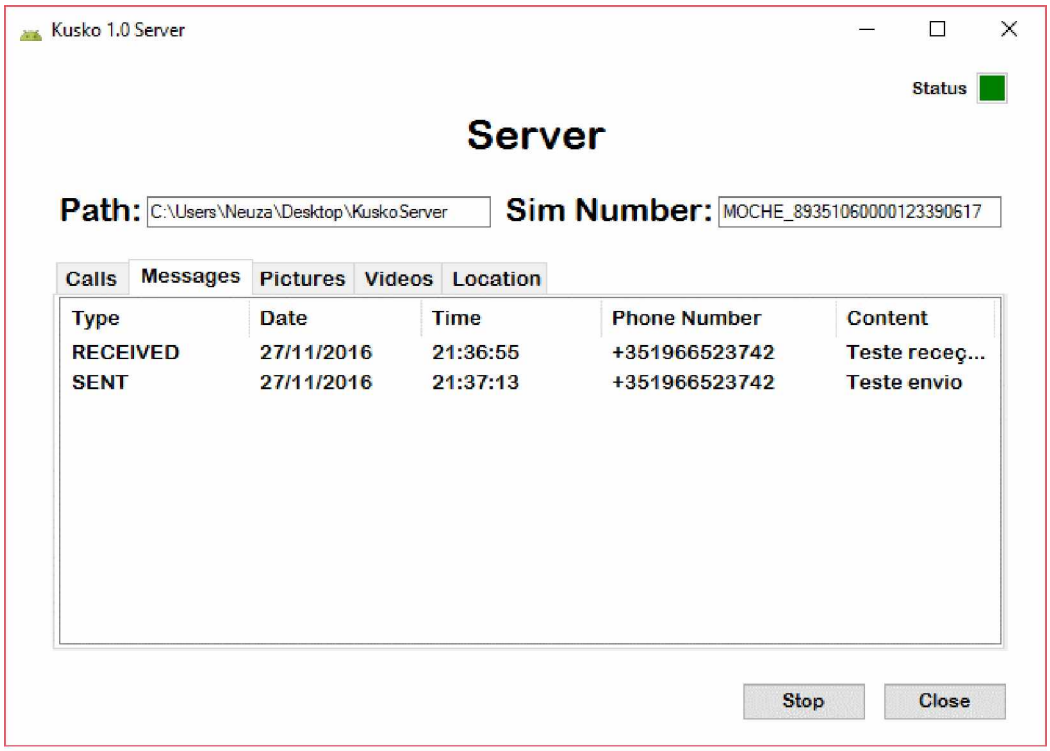

Figura 37 - Resultado das mensagens de texto no servidor

Tal como no caso das chamadas, também o duplo clique numa das mensagens da lista permite a visualização da informação a ela relativa numa nova janela, como mostra a imagem seguinte:

Figura 38 - Visualização da informação relativa a uma mensagem de texto

4.2.1.3. Fotografias e Vídeos

No caso dos testes de obtenção e envio de fotografias e vídeo foram capturadas duas imagens, com e sem localização GPS, e um vídeo. As imagens seguintes mostram os dados desses ficheiros no *smartphone*:

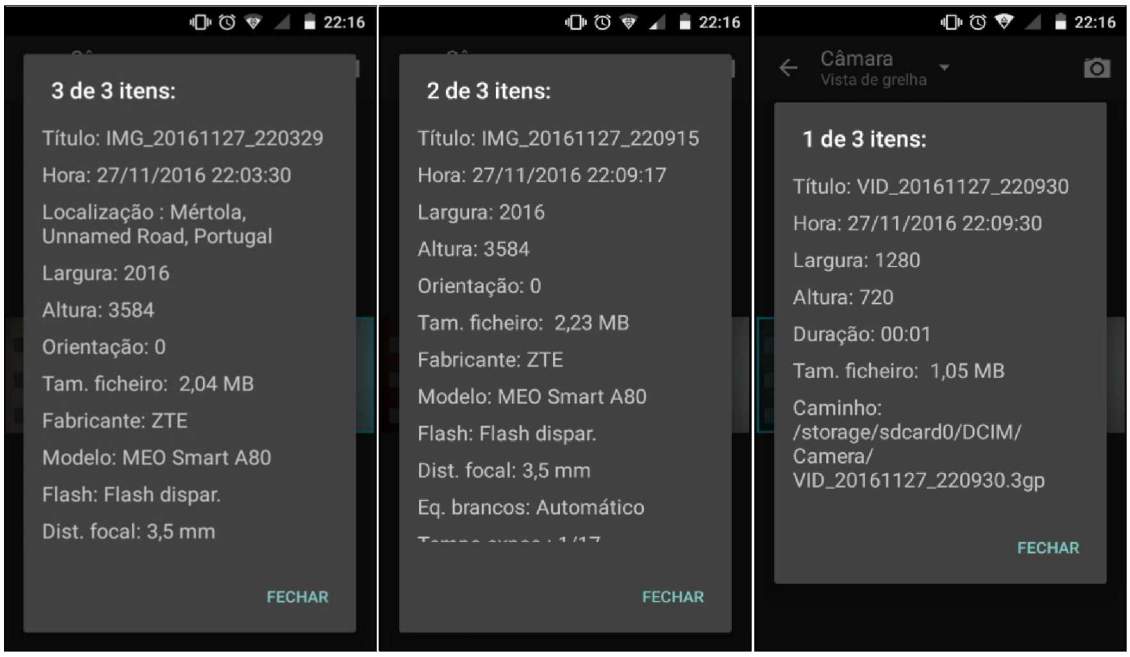

Figura 39 - Resultado das fotografias e vídeos no smartphone

Relativamente ao servidor, todas as fotografias e vídeos processadas pelo *smartphone* foram enviadas para este, como é possível verificar nas imagens seguintes:

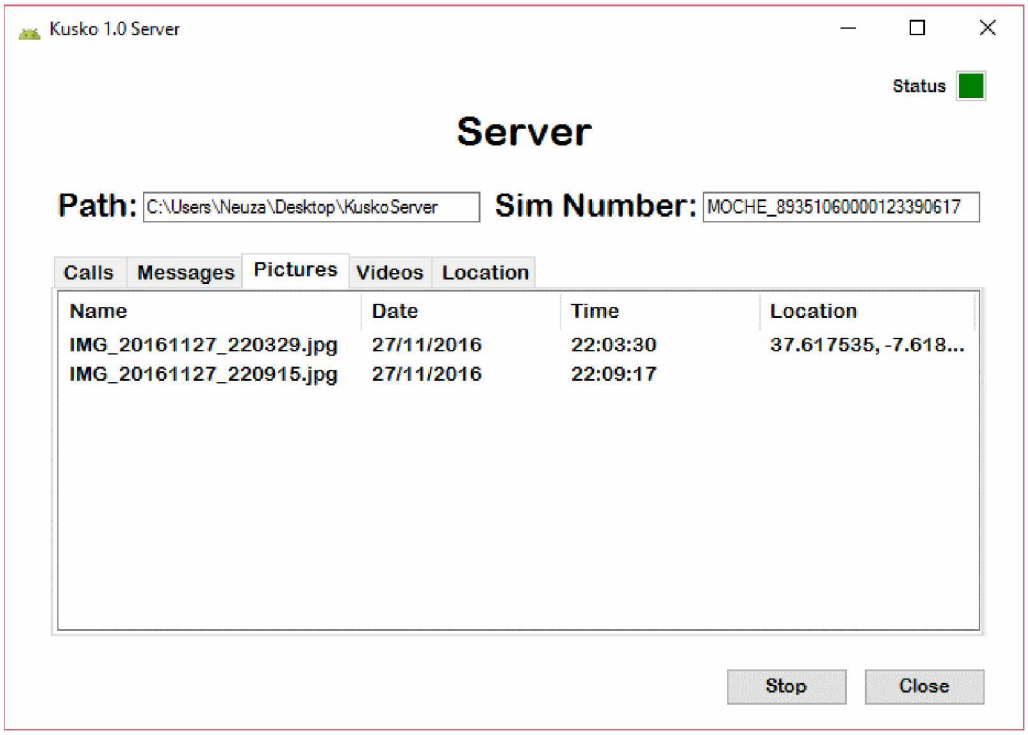

Figura 40 - Resultado das fotografias no servidor

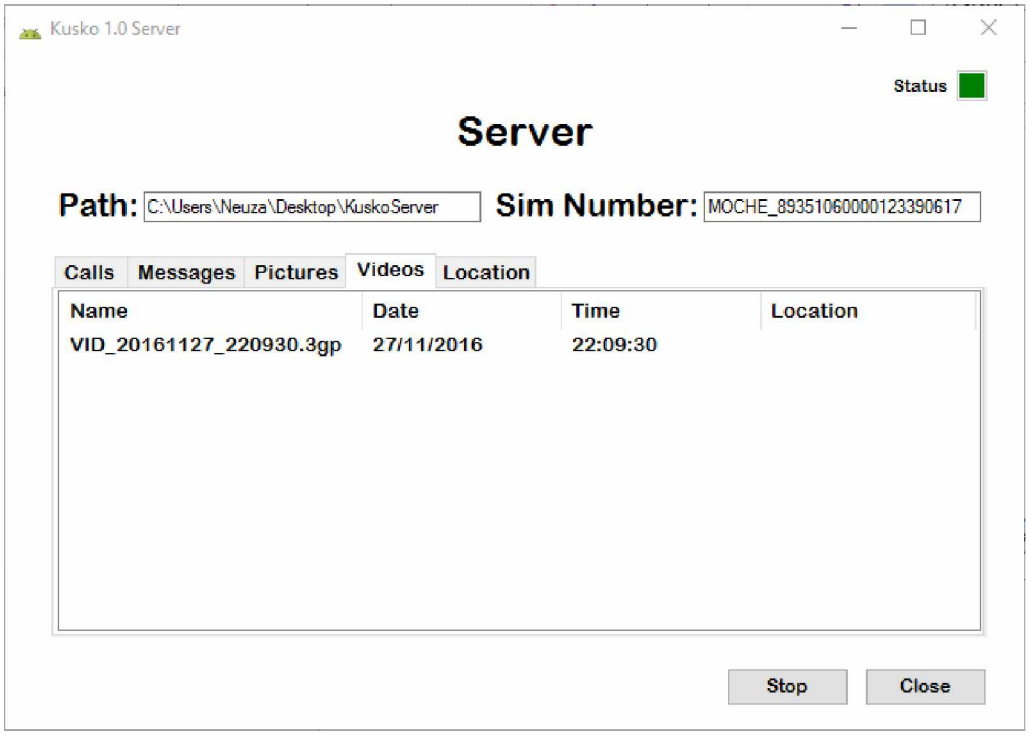

Figura 41 - Resultado dos vídeos no servidor

Tal como acontece com os ficheiros de áudio das chamadas, também os ficheiros das fotografias e vídeos se encontram nas pastas correspondentes, como mostra a imagem seguinte:

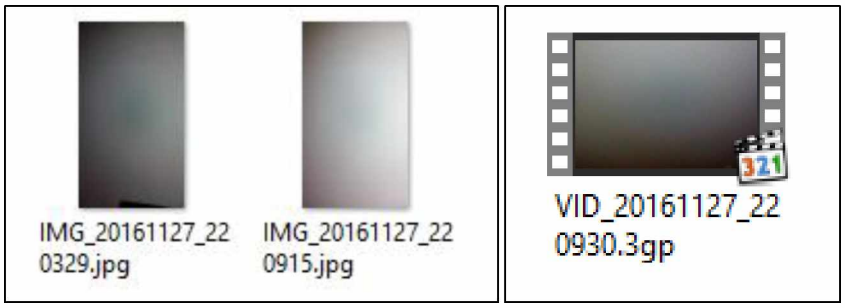

Figura 42 - Ficheiros das fotografias e vídeos

À semelhança das chamadas e mensagens, também no caso das fotografias e vídeos o duplo clique num dos itens da lista permite a visualização do ficheiro e da informação a ele relativa, como é possível observar nas imagens seguintes:

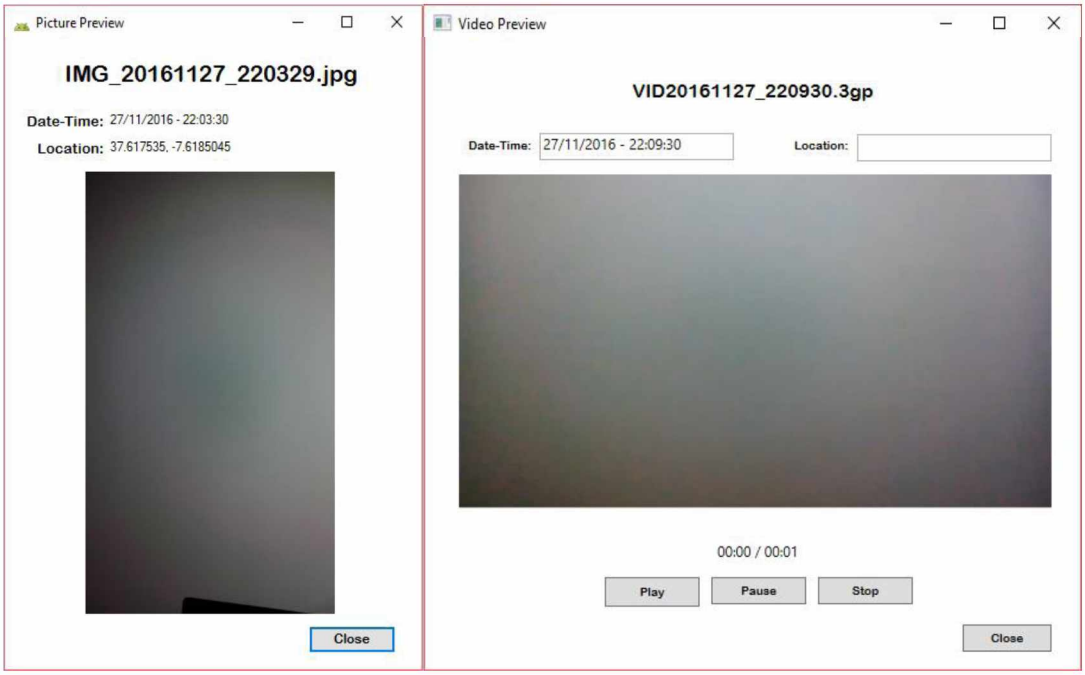

Figura 43 - Visualização das fotografias e vídeos

4.2.1.4. Localização

Por fim, o teste relativo à localização GPS do dispositivo obteve os resultados apresentados a seguir:

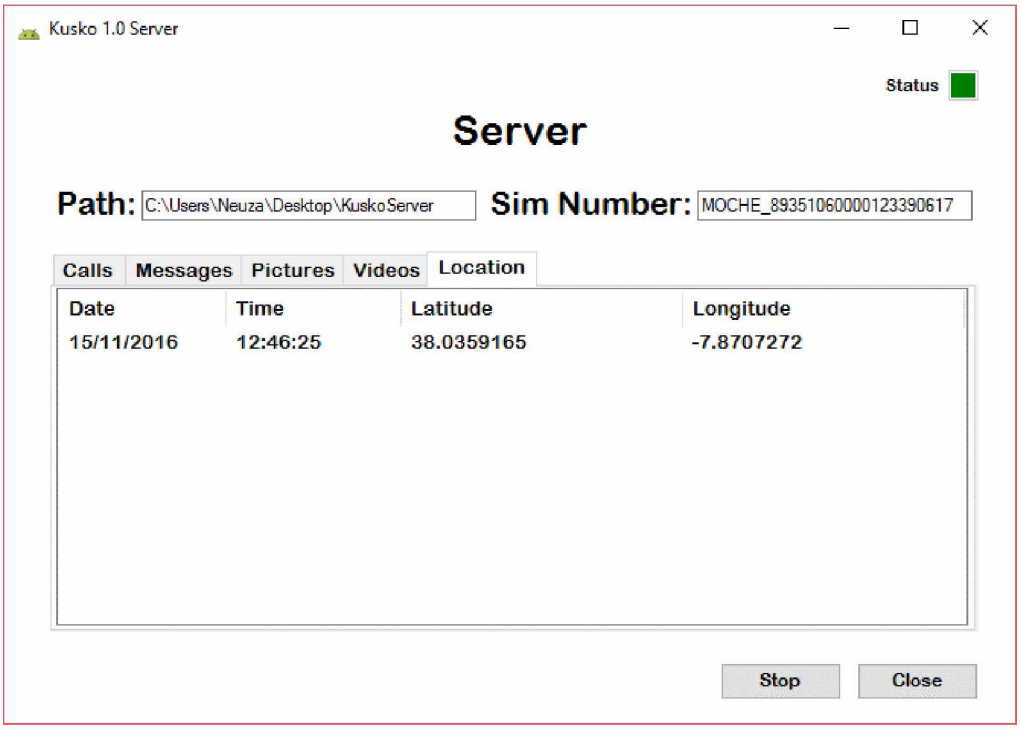

Figura 44 - Resultado da localização GPS no servidor

No caso da localização, o duplo clique num dos itens da lista abre uma nova janela que mostra a localização do dispositivo num mapa, conseguida através do recurso à API **GMap**. A janela em questão é apresentada a seguir:

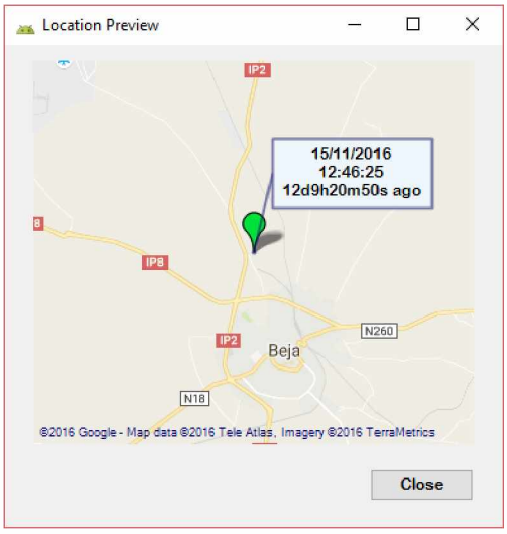

Figura 45 - Visualização dos dados da localização

4.2.1.5. Escrita de dados em ficheiro

Como referido anteriormente, quando o dispositivo móvel não possui ligação a uma rede sem fios, as capturas efetuadas são gravadas num ficheiro de texto para que possam ser enviadas para o servidor assim que existam condições para tal.

Para executar este teste, foi efetuada uma chamada de voz e enviada uma mensagem de texto. As imagens seguintes mostram a localização do ficheiro, juntamente com o ficheiro de áudio da chamada a enviar, e o conteúdo do ficheiro de texto:

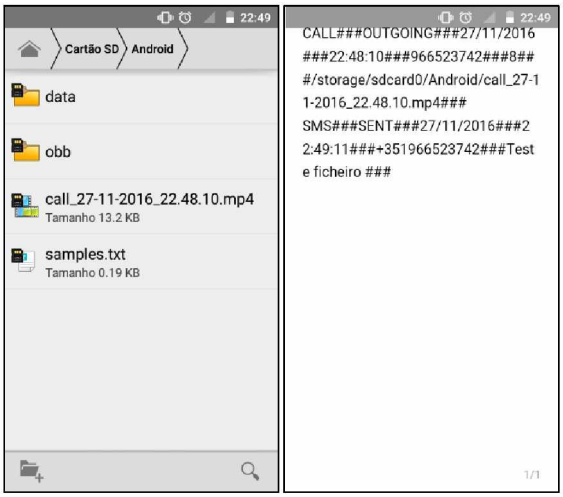

Figura 46 - Ficheiro de texto com capturas efetuadas

5. Conclusão e Trabalho Futuro

A execução desta dissertação exigiu um aprofundar de conhecimentos no que respeita ao sistema Android e às suas características de segurança e formas de ataque.

Foi também necessário explorar mais profundamente as linguagens de programação Java e C, as suas bibliotecas e funções, de forma a criar um código compacto e eficaz, tanto para o *malware* como para o servidor. Ainda que a criação e a ocultação de um *malware* sejam processos complexos são também bastante gratificantes.

Para a criação do *script* de ocultação do *malware* foi necessário efetuar um estudo das ferramentas necessárias para a criação e a modificação de APKs, e de todos os processos necessários para que o mesmo possa ser instalado num dispositivo. Ficam assim dados os primeiros passos no mundo dos testes de penetração e exploração de vulnerabilidades em sistemas Android.

Relativamente a trabalho futuro, seria de grande interesse a introdução de uma opção que permita a alteração remota do endereço IP do servidor incluído no código do *malware*, uma vez que atualmente o IP do servidor é estático e não pode ser alterado durante a execução do *malware*.

Essa alteração levaria à necessidade da encriptação desse endereço IP, caso o mesmo se encontrasse num ficheiro presente no dispositivo, por exemplo. Essa encriptação faria com que o endereço IP não fosse descoberto pelo utilizador, mesmo caso este encontrasse o ficheiro que o continha.

Por fim, seria ainda muito interessante estender o malware a outras aplicações de comunicação, como é o caso do **Messenger** do **Facebook** ou o **WhatsApp**, o que resultaria numa monitorização mais precisa e completa do dispositivo.

Bibliografia

[1] "**Tutorials Point - Android Architecture**," [Online].
Available: http://www.tutorialspoint.com/android/android_architecture.htm.
[Acedido em Julho 2016].

[2] J. Drake, Z. Lanier, C. Mulliner, P. O. Fora, S. A. Ridley e G. Wicherski, **Android Hacker's Handbook**, Indianapolis, IN: Wiley, 2015.

[3] "**Android Developer - App Components**," [Online].
Available: https://developer.android.com/guide/components/index.html.
[Acedido em Julho 2016].

[4] "**NXP - The Android Booting process**," [Online].
Available: https://community.nxp.com/docs/DOC-102546.
[Acedido em Julho 2016].

[5] N. Elenkov, Android Security Internals, An In-Depth Guide to Android's Security Architecture, San Francisco, CA: No Starch Press, 2014.

[6] "**Android Developer - Permissions**," [Online].
Available: https://developer.android.com/guide/topics/manifest/permission-element.html.
[Acedido em Julho 2016].

[7] "**Android Developer - Permission List**," [Online].
Available:https://developer.android.com/reference/android/Manifest.permission.html.
[Acedido em Julho 2016].

[8] "**Boundary Services-Android App Signatures & Permissions**," [Online].
Available: https://boundarydevices.com/android-security-part-1-application-signatures-permissions/.
[Acedido em Julho 2016].

[9] B. Rashidi e C. Fung, "**A Survey of Android Security Threats and Defenses**," 2015. [Online]. Available: http://isyou.info/jowua/papers/jowua-v6n3-1.pdf. [Acedido em Julho 2016].

[10] A. Lineberry, D. L. Richardson e T. Wyatt, "**These Aren't The Permissions You're Looking For**," 2010. [Online]. Available: https://www.defcon.org/images/defcon-18/dc-18-presentations/Lineberry/DEFCON-18-Lineberry-Not-The-Permissions-You-Are-Looking-For.pdf. [Acedido em Julho 2016].

[11] R. Fedler, C. Banse, C. Krauss e V. Fusenig, "**Android OS Security: Riskd And Limitations - A Practical Evaluation**," 2012. [Online]. Available:https://www.aisec.fraunhofer.de/content/dam/aisec/Dokumente/Publikationen/Studien_TechReports/deutsch/AISEC-TR-2012-001-Android-OS-Security.pdf. [Acedido em Julho 2016].

[12] Z. Lanier e A. Reiter, "**Mapping & Evolution of Android Permissions**," 2012. [Online].Available:http://www.countermeasure2012.com/presentations/LANIER_REITER.pdf. [Acedido em Julho 2016].

[13] "**Juniper Networks Mobile Threat Center - Third Annual Mobile Threats Report**," 2013. [Online]. Available: https://www.juniper.net/us/en/local/pdf/additional-resources/3rd-jnpr-mobile-threats-report-exec-summary.pdf. [Acedido em Agosto 2016].

[14] "**RisqIQ Reports 400% Increase Of Malicious Apps In Google Play Store**," 2014. [Online]. Available: http://thedroidguy.com/2014/02/riskiq-reports-400-increase-malicious-apps-google-play-store-85457. [Acedido em Agosto 2016].

[15] "**Pulse Secure Mobile Threat Center - Mobile Threat Report 2015**," 2015. [Online].Available:https://www.pulsesecure.net/download/pages/2819/PulseSecure_MobilityReport.pdf.
[Acedido em Agosto 2016].

[16] "**Android Malware Genome Project**," 2012. [Online].
Available: http://www.malgenomeproject.org.
[Acedido em Agosto 2016].

[17] Y. Zhou e X. Jiang, "**Dissecting Android Malware: Characterization and Evolution**," 2012. [Online].
Available: http://nieh.net/teaching/e6998/papers/OAKLAND12.pdf.
[Acedido em Julho 2016].

[18] "**Google Play Store**,"[Online].Available:https://play.google.com/store?hl=pt_BR.
[Acedido em Agosto 2016].

[19] Y. Zhou e X. Jiang, "**An Analysis of the AnserverBot Trojan**," Setembro 2011. [Online].Available:https://www.csc.ncsu.edu/faculty/jiang/pubs/AnserverBot_Analysis.pdf.
[Acedido em Agosto 2016].

[20] "**Symantec -BgServ**,"[Online].
Available:https://www.symantec.com/security_response/writeup.jsp?docid=2011-031005-2918-99&tabid=2.
[Acedido em Agosto 2016].

[21] "**Webopedia - DroidDream**," [Online].
Available: http://www.webopedia.com/TERM/D/droiddream.html.
[Acedido em Agosto 2016].

[22] "**Symantec - BaseBridge**," [Online].
Available:https://www.symantec.com/security_response/writeup.jsp?docid=2011-060915-4938-99&tabid=2.
[Acedido em Agosto 2016].

[23] "**F-Secure - DroidKungFu**," [Online]. Available: https://www.f-secure.com/v-descs/trojan_android_droidkungfu_c.shtml.
[Acedido em Agosto 2016].

[24] "**F-Secure - Plankton**," [Online]. Available: https://www.f-secure.com/v-descs/trojan_android_plankton.shtml.
[Acedido em Agosto 2016].

[25] "**Symantec - GGTracker**," [Online].
Available:https://www.symantec.com/security_response/writeup.jsp?docid=2011-062208-5013-99&tabid=2.
[Acedido em Agosto 2016].

[26] "**Symantec - Jifake**," [Online].
Available:https://www.symantec.com/security_response/writeup.jsp?docid=2012-073021-4247-99&tabid=2.
[Acedido em Agosto 2016].

[27] "**Symantec -Spitmo**," [Online].
Available:https://www.symantec.com/security_response/writeup.jsp?docid=2011-091407-1435-99&tabid=2.
[Acedido em Agosto 2016].

[28] "**SecureList - Zitmo**," [Online].
Available: https://securelist.com/analysis/publications/36424/zeus-in-the-mobile-facts-and-theories/.
[Acedido em Agosto 2016].

[29] "**NC State University - RogueSPPush**," [Online].
Available: https://www.csc.ncsu.edu/faculty/jiang/RogueSPPush/.
[Acedido em Agosto 2016].

[30] "**NC State University - DroidDeluxe**," [Online].
Available: https://www.csc.ncsu.edu/faculty/jiang/DroidDeluxe/.
[Acedido em Agosto 2016].

[31] "**Thesnkchrmr - RATC**," [Online].
Available: https://thesnkchrmr.wordpress.com/2011/03/24/rageagainstthecage/.
[Acedido em Agosto 2016].

[32] "**Symantec - Pjapps**," [Online].
Available:https://www.symantec.com/security_response/writeup.jsp?docid=2011-022303-3344-99&tabid=2.
[Acedido em Agosto 2016].

[33] "**NC State University - DroidKungFu3**," [Online].
Available: https://www.csc.ncsu.edu/faculty/jiang/DroidKungFu3/.
[Acedido em Agosto 2016].

[34] "**Symantec- Geinimi**," [Online].
Available:https://www.symantec.com/security_response/writeup.jsp?docid=2011-010111-5403-99&tabid=2.
[Acedido em Agosto 2016].

[35] "**Symantec - FakePlayer**," [Online].
Available:https://www.symantec.com/security_response/writeup.jsp?docid=2010-081100-1646-99.
[Acedido em Agosto 2016].

[36] "**AndroGuard - zSone**," [Online].
Available: http://androguard.blogspot.pt/2011/05/android-zsone-malware.html.
[Acedido em Agosto 2016].

[37] R. Balasubramanian, A. Modi e D. Pawade, "**Android Malware Detection: A Study and Comparison of the Most Promising Techniques**," Fevereiro 2016. [Online].Available:http://www.ijircce.com/upload/2016/february/87_41_Android.pdf.
[Acedido em Agosto 2016].

[38] G. Milbourne e A. Orozco, "**Android Malware Exposed: An In-depth Look at the Evolution of Android Malware**," 2012. [Online]. Available: https://www.webroot.com/shared/pdf/Android-Malware-Exposed.pdf. [Acedido em Julho 2016].

[39] "**ARSTechinica - New type of auto-rooting Android adware is nearly impossible to remove**," [Online]. Available:http://arstechnica.com/security/2015/11/new-type-of-auto-rooting-android-adware-is-nearly-impossible-to-remove/. [Acedido em Agosto 2016].

[40] "**LookOut Blog - A spike in Shedun, also known as HummingBad**," [Online]. Available: https://blog.lookout.com/blog/2016/07/06/shedun-hummingbad-hummer/. [Acedido em Agosto 2016].

[41] C. P. S. T. "**From HummingBad to Worse**," [Online]. Available:https://blog.checkpoint.com/wp-content/uploads/2016/07/HummingBad-Research-report_FINAL-62916.pdf. [Acedido em Agosto 2016].

[42] "**BetaNews - Android malware HummingBad generates $300,000 in monthly revenue**," [Online]. Available: http://betanews.com/2016/07/05/android-malware-hummingbad-monthly-revenue/. [Acedido em Agosto 2016].

[43] "**Spreitzenbarth - Current Android Malware**," [Online]. Available: http://forensics.spreitzenbarth.de/android-malware/. [Acedido em Agosto 2016].

[44] "**NetMarketShare-Mobile/Tablet Operating System Market Share**,"[Online]. Available: https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1&qptimeframe=Y. [Acedido em Agosto 2016].

[45] "**Statista - Android version market share distribution**," [Online]. Available: http://www.statista.com/statistics/271774/share-of-android-platforms-on-mobile-devices-with-android-os/.
[Acedido em Agosto 2016].

[46] "**The Verge - Android is now used by 1.4 billion people**," [Online]. Available: http://www.theverge.com/2015/9/29/9409071/google-android-stats-users-downloads-sales.
[Acedido em Agosto 2016].

[47] "**Mashable-iPhone Versus Android, Nielsen Delivers the Numbers**," [Online]. Available:http://mashable.com/2010/11/02/iphone-android-nielsen/#G5CVQnNPWkqb.
[Acedido em Agosto 2016].

[48] "**Android Authority - Does the average user care about Android versions?**," [Online]. Available: http://www.androidauthority.com/vicious-about-versions-how-much-do-we-really-care-about-android-updates-661529/.
[Acedido em Agosto 2016].

[49] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin e D. Wagner, "**Android Permissions: User Attention, Comprehension, and Behavior**," [Online]. Available:http://www.guanotronic.com/~serge/papers/soups12-android.pdf.
[Acedido em Agosto 2016].

[50] "**Android Developer - ContentObserver**," [Online]. Available:https://developer.android.com/reference/android/database/ContentObserver.html.
[Acedido em Agosto 2016].

[51] "**Android Developer - MediaRecorder**," [Online]. Available:https://developer.android.com/reference/android/media/MediaRecorder.html.
[Acedido em Agosto 2016].

[52] "**Android Developer - BroadcastReceiver**," [Online]. Available:https://developer.android.com/reference/android/content/BroadcastReceiver.html. [Acedido em Agosto 2016].

[53] "**Android Developer - FileObserver**," [Online]. Available: https://developer.android.com/reference/android/os/FileObserver.html. [Acedido em Agosto 2016].

[54] "**GitHub - RecursiveFileObserver**," [Online]. Available:https://github.com/owncloud/android/blob/master/src/com/owncloud/android/utils/RecursiveFileObserver.java. [Acedido em Agosto 2016].

[55] "**How To Geek - What is Exif Data and How to Remove It**," [Online]. Available:http://www.howtogeek.com/203592/what-is-exif-data-and-how-to-remove-it/. [Acedido em Setembro 2016].

[56] "**GPSTracker**,"[Online].Available:http://www.androidhive.info/2012/07/android-gps-location-manager-tutorial/. [Acedido em Setembro 2016].

[57] "**Android Developer - LocationListener**," [Online]. Available:https://developer.android.com/reference/android/location/LocationListener.html. [Acedido em Outubro 2016].

[58] "**Bulb Security - Backdooring APKs Programmatically**," [Online]. Available:http://www.bulbsecurity.com/backdooring-apks-programmatically/. [Acedido em Outubro 2016].

[59] "**Apktool**," [Online]. Available: https://ibotpeaches.github.io/Apktool/. [Acedido em Outubro 2016].

[60] "**Quora - Smali Code**," [Online]. Available: https://www.quora.com/What-is-smali-in-Android.
[Acedido em Outubro 2016].

[61] "**Docs Python - The ElementTree XML API**," [Online].
Available: https://docs.python.org/2/library/xml.etree.elementtree.html.
[Acedido em Outubro 2016].

[62] "**Docs Python - Regular Expression Operations**," [Online].
Available: https://docs.python.org/2/library/re.html.
[Acedido em Outubro 2016].

[63] "**Apache Ant**," [Online]. Available: http://ant.apache.org.
[Acedido em Outubro 2016].

[64] "**Oracle - JarSigner**," [Online].
Available:http://docs.oracle.com/javase/6/docs/technotes/tools/windows/jarsigner.html.
[Acedido em Outubro 2016].

[65] "**CoderWall - Generate Release/Debug Keystores**," [Online].
Available:https://coderwall.com/p/r09hoq/android-generate-release-debug-keystores.
[Acedido em Outubro 2016].

[66] "**Android Developer - Zipalign**," [Online].
Available:https://developer.android.com/studio/command-line/zipalign.html.
[Acedido em Outubro 2016].

[67] "**APK Downloader**,"[Online].Available:https://apps.evozi.com/apk-downloader/.
[Acedido em Novembro 2016].

[68] "**Google Play Store - Duck Duck Go Search & Stories**," [Online].
Available:https://play.google.com/store/apps/details?id=com.duckduckgo.mobile.android&hl=pt_PT.
[Acedido em Novembro 2016].

[69] "**VirusTotal**," [Online]. Available: https://www.virustotal.com/en/.
[Acedido em Novembro 2016].

[70] "**Bkav**," [Online]. Available: https://www.bkav.com.
[Acedido em Novembro 2016].

[71] "**Google Play Store - Instagram**," [Online].
Available:https://play.google.com/store/apps/details?id=com.instagram.android&hl=pt_PT.
[Acedido em Novembro 2016].

[72] "**Google Play Store - Opera Mini**," [Online].
Available:https://play.google.com/store/apps/details?id=com.opera.mini.native&hl=pt_PT.
[Acedido em Novembro 2016].

[73] "**Google Play Store - Candy Crush Saga**," [Online].
Available:https://play.google.com/store/apps/details?id=com.king.candycrushsaga&hl=pt_PT.
[Acedido em Novembro 2016].

[74] "**Google Play Store - Arrow Launcher**," [Online].
Available:https://play.google.com/store/apps/details?id=com.microsoft.launcher&hl=pt_PT.
[Acedido em Novembro 2016].

Anexo I – Amostras do Android Malware Genome Project

Malware	N.º Amostras	Android *Market*	Outros Mercados
FakePlayer	6		√
GPSSMSSpy	6		√
TapSnake	2	√	
SMSReplicator	1	√	
Geinimi	69		√
ADRD	22		√
Pjapps	58		√
BgServ	9		√
DroidDream	16	√	√
Walkinwat	1		√
zHash	11	√	√
DroidDreamLite	46	√	√
Endofday	1		√
Zsone	12	√	√
BaseBridge	122		√
DroidKungFu1	34		√
GGTracker	1		√
jSMSHider	16		√
Plankton	11	√	
YZHC	22	√	√
Crusewin	2		√
DroidKungFu2	30		√
GamblerSMS	1		√
GoldDream	47		√
HippoSMS	4		√
Lovetrap	1		√
Nickyspy	2		√
SndApps	10	√	
Zitmo	1	√	√
CoinPirate	1		√
DogWars	1		√
DroidKungFu3	309		√
GingerMaster	4		√
NickyBot	1		√
RogueSPPush	9		√
AnserverBot	187		√
Asroot	8	√	√
DroidCoupon	1		√
DroidDeluxe	1		√
Gone60	9	√	
Spitmo	1		√
BeanBot	8		√
DroidKungFu4	96	√	√
DroidKungFuSapp	3		√
DroidKungFuUpdate	1	√	√
FakeNetflix	1		√
Jifake	1		√
KMin	52		√
RogueLemon	2		√
Total	**1260**	**14**	**44**

Tabela 5 - Amostras de Malware do Android Malware Genome Project